Multi-Objective UAV Mission Planning Using Evolutionary Computation

THESIS

Adam J. Pohl, Second Lieutenant, USAF

AFIT/GE/ENG/08-22

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GE/ENG/08-22

Multi-Objective UAV Mission Planning Using Evolutionary Computation

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Adam J. Pohl, B.S.E.E.

Second Lieutenant, USAF

March 2008

AFIT/GE/ENG/08-22

Multi-Objective UAV Mission Planning Using Evolutionary Computation

Adam J. Pohl, B.S.E.E.
Second Lieutenant, USAF

Approved:

/signed/	6 Mar 2008
Dr. Gary Lamont (Chairman)	date
/signed/	6 Mar 2008
Dr. Gilbert Peterson (Member)	date
/signed/	6 Mar 2008
Maj. Michael J. Veth, PhD (Member)	date

AFIT/GE/ENG/08-22

Abstract

This investigation purports to develop a new model for multiple autonomous aircraft mission routing. Previous research both related and unrelated to this endeavor have used classic combinatoric problems as models for Unmanned Aerial Vehicle (UAV) routing and mission planning. This document presents the concept of the Swarm Routing Problem (SRP) as a new combinatorics problem for use in modeling UAV swarm routing, developed as a variant of the Vehicle Routing Problem with Time Windows (VRPTW). The SRP removes the single vehicle per target restraint and changes the customer satisfaction requirement to one of vehicle on location volume. The impact of these alterations changes the vehicle definitions within the problem model from discrete units to cooperative members within a swarm. This represents a more realistic model for multi-agent routing as a real world mission plan would require the use of all airborne assets across multiple targets, without constraining a single vehicle to a single target. Solutions to the SRP problem model result in route assignments per vehicle that successfully track to all targets, on time, within distance constraints. A complexity analysis and multi-objective formulation of the VRPTW indicates the necessity of a stochastic solution approach leading to the development of a multi-objective evolutionary algorithm. This algorithm design is implemented using C++ and an evolutionary algorithm library called Open Beagle. Benchmark problems applied to the VRPTW show the usefulness of this solution approach. A full problem definition of the SRP as well as a multi-objective formulation parallels that of the VRPTW method. Benchmark problems for the VRPTW are modified in order to create SRP benchmarks. These solutions show the SRP solution is comparable or better than the same VRPTW solutions, while also representing a more realistic UAV swarm routing solution.

Acknowledgements

I would like to acknowledge the massive amount of help I have received from my peers here at AFIT. Specifically, thanks to Dan Karrels who allowed me to bypass my inability to compile and write C++ code in the early and later stages of this project. Everything I know about programming and Unix up to this point I attribute to him. Thanks to Dustin Nowak for explaining different aspects of evolutionary computation and providing a sounding board for various topics. Thanks to John McCall for reading my thesis, giving me feedback, and of course, for keeping it real. Thanks to my Dad for reading my thesis in the early stages and providing feedback.

I owe great appreciation for my wife supporting me in my work and helping me deal with the stresses and strain of the previous year. She has kept me alive for the past few years to the benefit or detriment of those around me.

Credit to my advisor Dr. Lamont for allowing me to freely pursue my ideas and find my own way through this process. Also, thank you for always being understanding even with the many mistakes and mis-steps made along the way.

I am hopeful that this and future work will contribute to my overarching goals for humanity, *Neca Eos Omnes Humanaeas*.

Adam J. Pohl

Table of Contents

Page

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

GA	Genetic Algorithm
GP	Genetic Programming
EA	Evolutionary Algorithms
EC	Evolutionary Computation
TSP	Traveling Salesman Problem
MOEA	Multi-objective Evolutionary Algorithm
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows
SRP	Swarm Routing Problem
UAV	Unmanned Aerial Vehicle
AUAV	Autonomous Unmanned Aerial Vehicles
GVR	Genetic Vehicle Representation
PSO	Particle Swarm Optimization
NSGA2	Non-dominating Sorting Genetic Algorithm 2
SPEA2	Strength Pareto Evolutionary Algorithm 2
PDES	Parallel Discrete Event Simulation
SO	Self Organized
OB	Open Beagle
AFIT	Air Force Institute of Technology
OO	Object Oriented
PBS	Portable Batch System
USAF	United States Air Force

Multi-Objective UAV Mission Planning Using Evolutionary Computation

I. Introduction

This research investigation studies the routing of multiple UAVs and UAV swarms to a set of locations while meeting constraints of time on target, total mission time, enemy radar avoidance, and total path cost optimization. United States Air Force (USAF) research has been focused, and increasingly continues to focus, on the development of Autonomous Unmanned Aerial Vehicles (AUAV). The development of UAV technology is already advanced, with uses being found both for reconnaissance and tactical strikes [52].

While hardware for these UAV systems continues to evolve, cutting edge research focuses on the development of taking unmanned systems and using autonomous control schemes with only a high level strategic decision maker (human) in the loop. Developing a single autonomous UAV is not the objective, rather the objective is to develop a massive array of AUAV, capable of working together toward a common goal. The term for this array is a swarm or flock for which there are many different design approaches as the problem itself exists in many scientific and engineering domains. Currently, this forefront technology exists only in simulation form as hardware and sensors capable of implementing an actual swarm simply do not exist. This research focuses on the development of off-line UAV routing and mission planning, combined with a simulation and visualization system the purpose of which is to better understand the computational complexity of multi AUAV routing.

1.1 Problem Statement

The problem of mission planning consists of assigning multiple vehicles sets of targets to visit. These targets exist in a field of uneven terrain where different enemy radar lines of sight exist. There exist two problem aspects to deal with, the first is

the development of flight paths between targets. The path must be optimized for cost and risk. Cost is how much energy or time it takes to traverse the path and risk is a measure of how dangerous the flight area is. The second is the development of path order. Once it is determined how to best fly between targets the order of these flight paths must be determined. Generating the cost of the path is a separate and immaterial problem related to the development of path order. In fact, the development of single path optimization is already being studied in fields such as robotics, land, and air based agents. Once these path costs are known, or estimated, however what development process should be used to structure a valid route plan from them? This process of route development is the subject matter of this investigation.

In order to model this routing situation, a combinatorics problem known as the Vehicle Routing Problem with Time Windows (VRPTW) is used. The VRPTW encompass a situational problem composed of a number of vehicles, known targets with time visitation constraints, and constraint on the visitation capacity of each vehicle. This model most efficiently possesses all the aspects of the problem under consideration and is well documented and understood. The VRPTW is limited in its ability to model realistic UAV routing, necessitating in its extension into a new problem model. This VRPTW variation is called the Swarm Routing Problem (SRP), presented as a more efficient form to model the routing of multiple UAVs to multiple targets concurrently. This problem structure is illustrated in Figure 1.1. Note that the path planner supplies cost values to the router portion of the structure. What these values are is irrelevant to the router itself. The purpose of the router is to efficiently search the innumerable combinations of costs supplied to in order to determine the most efficient (or cost effective) path possible. The remainder of this section introduces these problem concepts in greater detail.

1.1.1 The Vehicle Routing Problem with Time Windows. The Vehicle Routing Problem (VRP) is a classic combinatorics problem in the field of computer science and the VRPTW is a variant of it. In this context it models the routing of UAVs

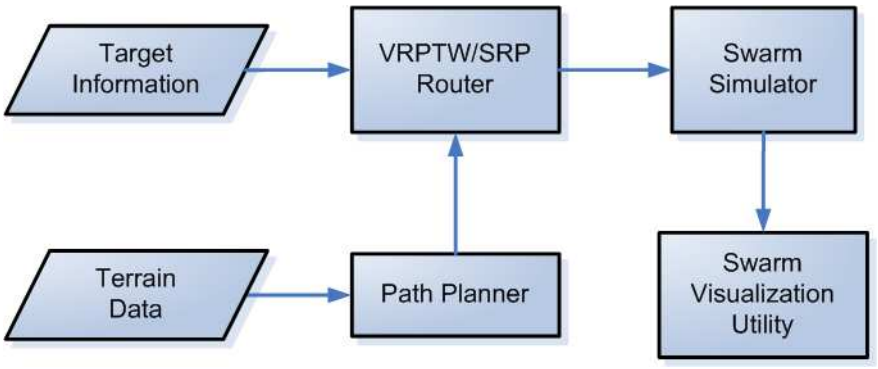

Figure 1.1: Problem Structure

to a set of targets within a time window for each target and an overall time limit for returning to the start location. The objective of the problem is to develop a set of routes for a fleet of vehicles where each vehicle services a target within its time window and returns to the start point after all the targets in its route have been serviced. The problem is constrained by how many targets a vehicle can visit, each target's time window, and a return time limit to the start location [51].

1.1.2 The Swarm Routing Problem. The SRP is a variation on the VRPTW defined for the first time in this work. While the classic VRPTW restricts each vehicle to a single route and target, the SRP assumes the vehicles can join together and split apart at different locations, in essence treating each vehicle as a member of a sub swarm as opposed to a discrete vehicle. The effect of using the SRP to model UAV swarm mission routing is that a more efficient and realistic deployment plan is created. Basing the model on the VRPTW allows for the use of previously made insights into the VRP and VRPTW to create effective solutions for the SRP. Analysis of the VRPTW also allows for the development of comparable results during the solution evaluation phase, the result of which is a metric of performance to compare the SRP results to. The SRP problem complexity is NP-Hard by merit of its derivation from the VRPTW. Chapter III contains a more exact complexity analysis.

1.2 Routing Model Assumptions

The routing model (for both VRPTW and SRP) is viewed as a static combinatorics problem. While the VRPTW model does take into account time constraints for targets that is also all it is capable of doing. Modeling the routing problem as a VRPTW imposes the limitation that the solution only be for the VRPTW. As such, no dynamic aspects can be introduced such as in-mission rerouting or defensive measures. The problem does not take possible losses of aircraft during the mission into account. This limits the usefulness of the model as a real world problem solver, but is a closer approximation than a more simplistic point to point router without time constraints.

The VRPTW and SRP are also generalized combinatorics problems and no attempt is made integrate real world problem constraints such as vehicle movement constraints, real world distance constraints, or communication issues. The solution to these general models represents a top level routing strategy where the actual values of individual route sections, or how they are determined, is immaterial. In reference to the routing problem, there are only the individual path cost values, what they actually are or how they are developed exists only within the users perspective.

1.3 Research Concepts

The research concepts developed as part of this investigation involved the following solution approaches. They are introduced to give the reader a better understanding of the direction taken in this endeavor.

1.3.1 Multi-objective Evolutionary Algorithms. Due to the high complexity of the problem domain in this investigation, an evolutionary algorithm approach is used as previous work that has shown a great deal of success through its utilization [1, 33, 40, 44, 60]. Evolutionary algorithms encompass a number of different stochastic search techniques where random solutions are "bred" into more effective solutions through a process of selection and mutation. This work examines how multi-objective

approaches obtain the most effective results for highly constrained solution spaces, such as those dealt with in this research. The concept of multi-objective evolutionary search, non-dominated (Pareto) front solutions, as well as alternative methods of multi-objective optimization is further examined in Chapter II.

1.4 Research Goals, Objectives, and Assumptions

The goal of this research centers on the creation of new routing software for the UAV swarm routing problem developed in previous research [11, 44]. The solution is designed in the context the it integrate into previous work in path planning development and UAV simulation.

1.4.1 Objective 1: Develop SRP Multi-objective Problem Model. The first objective is to develop a problem model for multiple UAV routing. The VRPTW serves as a basis for the development of this model. The purpose in this is to create a problem that serves as a more realistic model for developing time constrained routes to multiple targets. The problem is formulated in terms of the multiple objectives of total path length, vehicle count, vehicle wait time, and average path length.

1.4.2 Objective 2: Develop and validate MOEA solution to VRPTW. The second objective is to develop a multi-objective evolutionary solution to the VRPTW. This is accomplished by examining previous MOEA approaches to the VRP and VRPTW in order to develop effective genetic operators for a new implementation. Completion of this objective consists algorithm development and fully defining genetic operators the Genetic Algorithm (GA) uses. The purpose in developing a solution to the VRPTW is to obtain a reference point for development of the SRP. The SRP represents an untested model, therefore the most intelligent action is to validate the solution procedure on a pre-existing problems.

A variety of different benchmark problems are solved using the implemented algorithm and the results are compared to those found in the literature. The com-

pletion of this objective is signified by the MOEA approach achieving as good or better results than those found in the literature and a documentable distribution of solutions across the objective landscape. The objectives of comparison are total path length, vehicles used, and total wait time for all vehicles. Each of these objectives then represent metrics of performance meant to be minimized. Experimental analysis consists of a variety of algorithmic approaches in order to examine the utility of the each method and statistical analysis is used to show the significance of the different methods used.

1.4.3 Objective 4: Develop and validate MOEA solution to SRP. The forth objective is to develop a solution to the SRP. This new problem is a variation of the VRPTW and is solved in a similar fashion. A set of benchmark problems are created by altering existing VRPTW problems. These benchmark problems are then solved using a solution designed parallel to the one developed for the VRPTW.

1.5 UAV Model Assumptions

No attempt is made to completely model the aerodynamics of the UAVs used. They are assumed to be point masses whose control scheme is some undefined method, though the scheme would most likely be some form of self-organized control. All routing operations take place off-line. It is assumed that the UAVs have some form of communication method though no attempt is made in this work to model the type of communication architecture the swarm may use. It is assumed control and sensor mechanisms exist that allow the UAVs to pilot themselves without the need for human management.

References throughout are made to UAVs. The term is meant to refer to autonomous UAVs (AUAV), not remote controlled vehicles. Chapter II contains background information on self organized control theory which is presented as a possible control method for AUAVs traversing through a routing solution.

1.6 Sponsor

This research is sponsored by the Air Force Research Laboratory (AFRL) Sensors Applications and Demonstrations Division (AFRL/SNZ), specifically, the Virtual Combat Laboratory (AFRL/SNZW) at Wright Patterson Air Force Base, Ohio. The Virtual Combat Laboratory conducts advanced development initiative as well as field and flight test demonstrations and evaluations. A series of UAV simulation models are maintained within the lab along with a suite of visualization tools. This research continues the ongoing relationship between AFIT/ENG and AFRL/SNZW in which both parties share information on, and enhance the capabilities of, UAV swarm simulations. The point of contact at AFRL is Mr. Mike Foster (AFRL/SNZW).

1.7 Thesis Construction

This thesis document has been constructed with the goal that it be readable, efficient and thorough. The basic concepts of research are listed in this chapter and expanded upon in Chapter II. A thorough examination of the research problem under consideration is presented in Chapter III. High level solution composition is detailed in Chapter IV. The implementation documentation is contained in Chapter V and Appendix B. Chapter VI details the experiments used to validate the solution techniques proposed. Chapter VII contains results and analysis. Chapter VIII concludes the thesis with a discussion of contributions suggested future endeavors.

II. Background Research

This chapter is divided into several sections relevant to the areas of research pursued. The first section discusses previous work that this research extends. Each subsequent section is presented as an examination of relevant background information. The background information also contains problem domain formulations of the vehicle routing and path planning problems. The second section is an analysis of the Vehicle Routing Problem with Time Windows (VRPTW) and how its solution can be used to effectively solve the Unmanned Aerial Vehicle (UAV) routing problem for multiple targets. The third section is an examination of the path following constraint that is formulated in conjunction with the VRPTW. The fourth section covers the topic of Multi-objective Evolutionary Algorithm (MOEA)s. Pareto ranking techniques are discussed as well as the benefits of using MOEA approaches over weighted and single objective techniques. The chapter concludes with a section on the concept of self organized UAV swarms. Previous work in this area as well as its context in this work is examined.

2.1 AUAV Simulation Research History at AFIT

Research into the problem of effectively routing and controlling a swarm of UAVs has been under active development since 2000 (at AFIT). The problem is by no means simple and crosses into many scientific fields. Currently, the technology exists mainly in the simulation arena, though much work has been done with single UAV control [1]. The following discussion introduces previous thesis research efforts that have dealt with the topic of UAV routing and simulation. This information is summarized in Table 2.1.

Research originally began with UAV routing by modeling the problem as a classic combinatoric problem called the Traveling Salesman Problem (TSP). In Sezer [43], an advanced search tree method was developed for solving a constrained mission planning problem. Research then moved into stochastic methods when Secrest [42] developed a particle swarm optimization method for solving the TSP. At this point it had

Table 2.1: Previous AFIT theses related to AUAV routing.

Title	Author	Year	Subject
Mission Route Planning with Multiple Aircraft & Targets Using Parallel A* Algorithm	Ergin Sezer	2000	Using a tree search method to solve routing problems
Traveling Salesman Problem for Surveillance Mission Using Particle Swarm Optimization	Barry Secrest	2001	Using a combinatorics problem as a model for UAV routing
Multi-objective Mission Route Planning Using Particle Swarm Optimization	Ekursat Yavuz	2002	Using PSO search method to route UAVs
Distributed Control of a Swarm of Autonomous Unmanned Aerial Vehicles	James Lotspeich	2003	Examining the control problem of UAV swarms
Swarming Reconnaissance Using Unmanned Aerial Vehicles In A Parallel Discrete Event Simulation	Joshua Corner	2004	Developing the AFIT swarm simulator
Evaluation and Optimization of UAV Swarm Multi-Objectives	Mark Kleeman	2004	Developing the multi-objective idea of UAV routing
Evolution of Control Programs for a Swarm of Autonomous Unmanned Aerial Vehicles	Keven Milam	2004	Using GP to develop the control structure
A Genetic Algorithm for Unmanned Aerial Vehicle Routing	Matt Russell	2005	Applied new chromosome structure to the VRP
Evolving Self Organizing Behavior for Homogeneous and Heterogeneous Swarms of UAVs and UCAVs	Ian C. Price	2006	Optimizing the control structure of UAVs
AFIT UAV Swarm Mission Planning and Simulation System	James Slear	2006	Development of path planning software

been established that the high complexity of the routing problem necessitated the use of stochastic methods in order to achieve results approaching optimality in a tractable time frame. In Yavuz [59], particle swarm optimization was used to develop optimal solutions to a multi-objective routing problem. Based on lessons learned in that thesis effort (of the different objectives relevant to the problem), further work was done by Russell [40] where the problem was formulated as a VRP and solved using a GA [7]. Slear [44] used this problem formulation while path planning functionality was added.

In Kadrovach [23], Lotspeich [28], and Milam [31] methods for Self Organized (SO) control were developed. These control schemes were based on flock and swarm behaviors first defined by Reynolds [38]. Simulations using these control schemes were at first, two dimensional but illustrated the validity of the idea behind flocking and swarming.

After it was shown that self organized control could be effectively implemented, research diverged into three areas: swarm routing, swarm communication, and SO algorithm development. Swarm communication was approached by Kadrovach [23] and Kleeman [24] where it was shown that adequate communication within a real world swarm is an extremely complex problem. Disregarding the hardware limitations of current wireless communication technology and the number of external sensors that would be required, simply developing the algorithms that would be able to handle the massive amount of internal swarm communication is still a largely unsolved problem. Current developments in self organized control have centered on creating MOEAs which optimize the SO rule settings. Price [36] developed an evolutionary algorithm and simulator that was able to optimize attack patterns of the swarm in a 2D simulation. Different attack patterns as well new optimization strategies were created by Nowak [32].

Both Russell [40] and Slear [44] utilized the AFIT UAV simulator which was first developed by Kadrovach [23] and then ported into a Linux environment and parallelized in Corner [11]. Corner [11] also developed a Parallel Discrete Event Simulation (PDES) methodology to simulate larger and more complex SO swarms in 3D.

Many different languages and methodologies have been applied to the development of routing, control and simulation solutions. The most popular coding languages (in related thesis work and in published research) have been Matlab, Java^{TM}, and C++. A 2D flocking simulation in Matlab was created by Watson [54] to study the dynamics of the self organizing behavior. A similar simulation developed in Java was

created by Price [36] which treated the UAVs as point masses with no dynamic control mechanisms. Previous routing [40] and path planning programs [44] have been created using C and C++ and both utilized an evolutionary computation library called GALib [53] (also written in C++), as the support library for their solution techniques. The simulator software is based on a C++ library called SPEEDES, which was developed to provide communication support for PDES. The selection of this library was based on an investigation completed by Corner [11].

The problem of what a swarm actually does when it reaches a target (i.e. mission type) has only recently been dealt with due to the routing problem itself being such a complex endeavor. Some recent work has been conducted in 2D environments that simulate different UAV missions [36]. Currently, UAV missions are viewed in a purely reconnaissance oriented role. With the capabilities of a real-world swarm there are no limits to what its capabilities would be. A swarm could be sent out to perform a concentrated attack on a target, patrol for and attack targets of opportunity, or even escort manned targets across hostile areas by not only targeting enemy units but patrolling the area to alter the course of the manned unit with real time data.

The approach to UAV swarm simulation has been very iterative in development, first with problem development and understanding, then routing and control solutions, and more recently advanced simulations. This research also serves as an iterative step to improve on the problem model used to develop routing solutions, apply true multi-objective techniques to the solving routing and path planning problem and enhance the current simulator technology. The basic problem remains the same now as it has always been:

Develop a method for taking a set of targets within an area of terrain and finding a set of paths for each UAV or UAV swarm such that all sites are visited by an appropriate number of UAVs to accomplish the desired mission while minimizing path cost and maximizing UAV safety.

2.2 Vehicle Routing Problem with Time Windows

The VRP is a well established combinatorics problem with many variations and solutions [51], one of these variations being the Vehicle Routing Problem with Time Windows (VRPTW) [51]. The problem is most easily explained with a real world example. There exists a package delivery service warehouse that has a number of trucks. Each truck can carry a limited load of packages to customers. Customers are spread out around the depot and each can only be visited during a certain time of the day. Likewise, the warehouse itself is only open for a certain number of hours until it closes. Determine the shortest set of paths for each vehicle to take, such that all customers are visited within their window of opportunity and all vehicles return on time.

From this example, it is seen that the VRPTW consists of a set of targets, some number of vehicles, and a depot. The depot is the deployment and return point for all the vehicles. Each target (and the depot) has a Euclidian location (coordinate), some associated demand (except the depot), an arrival time window, and a path to every other location. The objective of the problem is to develop a set of routes for each vehicle, so that all targets are visited within the time window, the associated demand is met, and all vehicles return to the depot on time. The solved problem is visualized in Figure 2.1 where the darker areas correspond to the time window in which a vehicle can visit the target. Each vehicle in the problem has a set capacity that it can not exceed while visiting customers. Visiting a customer subtracts its demand from the vehicles capacity. The total demand on the vehicle is the sum of the demands of all the customers visited. If the vehicles being used do not have some type of capacity constraint the problem then decomposes to a TSP since one vehicle can now satisfy all customers. The remainder of this section is a more detailed mathematical description of the single-objective VRPTW presented in order to better understand the constraints and optimization goal of the problem.

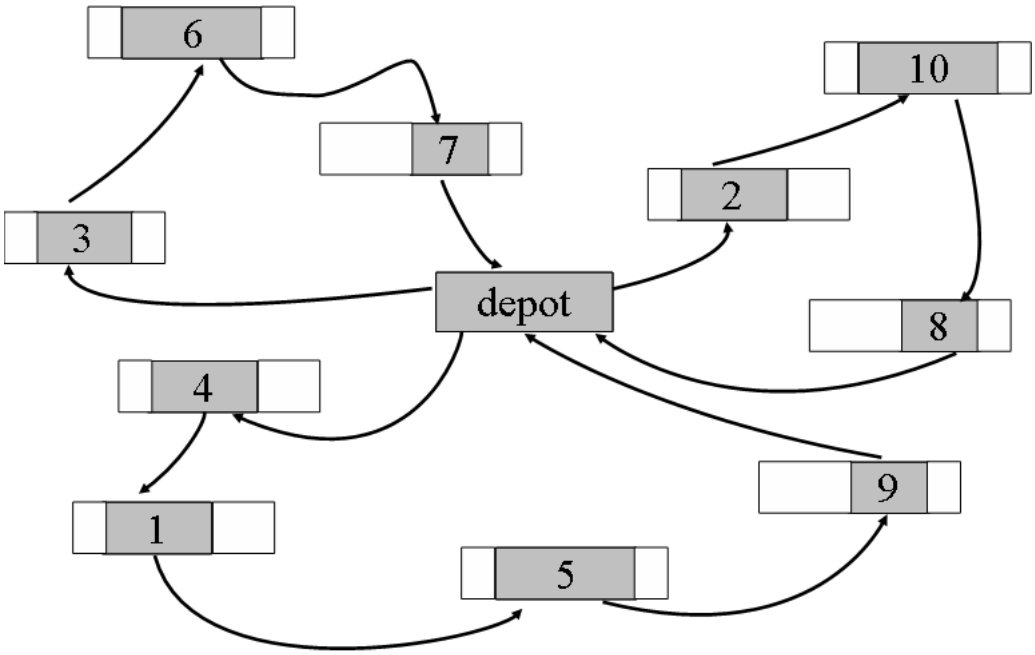

Figure 2.1: The Vehicle Routing Problem with Time Windows.

The VRPTW, as formulated by Toth [51] based on the ordinal formulation by Solomon [45], is defined by a fully connected graph $G = (V, A)$ where $V = \{v_0, v_1, ..., v_n\}$ and v_0 is the depot. The set of edges is defined as $A = \{(v_i, v_j) \in V, i \neq j\}$ where each edge has associated with it some travel cost $c(v_i, v_j)$. The cost of the edge is the distance from target i to target j in the context of the problem discussed here. Cost is not necessarily distance however, cost is only a value that is associated with traversing the route. It could be the distance, it could be the time, or it could be the amount of fuel required to travel the path. A time window exists for all customers where E represents the earliest start time and, L, the latest arrival time. The latest arrival time is the point at which the vehicle can arrive and still complete the service time defined by S. If the vehicle arrives earlier than E, it incurs a waiting time, W, which is the difference between the arrival time and E. The total time a vehicle takes to complete it's route is the summation of all route path travel costs, waiting times, and service times ($\sum c_{ij} + \sum w_i + \sum s_i$) and is refereed to as the cost of the path. The total path cost must not exceed the latest arrival time (i.e.

closing time) of the depot. Cost is, in essence, the total time required to traverse the route.

Each customer also has associated with it some demand, D, satisfied by the vehicle servicing it. There exists a number of vehicles, K, where each vehicle has some capacity constraint, C, such that the sum of the demands, D, for all customers visited by, $k \in K$, does not exceed C. The solution to the problem is a list of ordered targets for each vehicle, such that the visitation of each target fulfills all customer needs without violating any time or demand constraints. The objective then is to determine the set of paths for the vehicles such that the total cost is minimized. Note that the cost of a route is not the total time the route takes to complete, it is only the sum cost of the edges the vehicle traverses.

The following is a mathematical programming formulation of the single-objective VRPTW, again taken from [51]. Vehicles are defined within the problem by their inclusion in a flow variable x_{ijk}, which is a binary value indicating if vehicle k exists on the path that connects $(i, j) \in V$ at any point in the solution. A time variable ω_{ik} indicates the start time of vehicle k at location i. The subscript $j \in \Delta^{\pm}(i)$ indicates the set of edges from i to j where j is not equal to i, the plus or minus indicates either a forward or backward move along the path.

A_{ij} - Edge cost between i and j

V_n - Network vertices for n customer (v_0 is the depot)

E_n - Earliest arrival time of customer n

L_n - Latest arrival time of customer n

S_n - Service time of customer n

D_n - Demand of customer n

K - Set of Vehicles

C_k - Capacity of vehicle k

$$x_{ijk} \in \{0,1\} \qquad \forall k \in K, (i,j) \in A, \tag{2.1}$$

$$\omega_{ik} \geq 0 \qquad i \in N, k \in K, \tag{2.2}$$

Equations (2.1) and (2.2) define the flow and time variables used. The flow variable x_{ijk} is a binary value that indicates vehicle, k, travels from location, i to j, if equal to one, and zero otherwise. The time variable, ω_{ik}, specifies the start time at location, i, by vehicle k.

$$\sum_{k \in K} \sum_{j \in \Delta^+(i)} x_{ijk} = 1 \qquad \forall i \in N, \tag{2.3}$$

$$\sum_{j \in \Delta^+(0)} x_{0jk} = 1 \qquad \forall k \in K, \tag{2.4}$$

$$\sum_{i \in \Delta^-(n+1)} x_{i,n+1,k} = 1 \qquad \forall k \in K, \tag{2.5}$$

$$\sum_{i \in \Delta^+(j)} x_{ijk} - \sum_{i \in \Delta^-(j)} x_{ijk} = 0 \qquad \forall k \in K, \forall i \in N, \tag{2.6}$$

Equations (2.3)-(2.6) define the edge constraints of the graph in a solution. They indicate that each customer can only be assigned to a single route (2.3), that all vehicles with a route must start from the depot (2.4), that all edge costs are symmetrical (2.5), and that all vehicles must return to the depot (2.6).

$$x_{ijk}(\omega_{ik} + s_i + t_{ij} - \omega_{jk}) \leq 0 \quad \forall k \in K, (i,j) \in A, \tag{2.7}$$

$$e_i \sum_{j \in \Delta^+(i)} x_{ijk} \leq \omega_{ik} \leq l_i \sum_{j \in \Delta^+(i)} x_{ijk} \quad \forall k \in K, \forall i \in N, \tag{2.8}$$

$$e_i \leq \omega_{ik} \leq l_i \qquad \forall k \in K, i \in (0, n+1), \tag{2.9}$$

Equations (2.7)-(2.9) define the time constraints of the problem. Equation (2.7) indicates that the arrival time at location i plus the service time and travel time to the next location must equal the arrival time. Equation (2.8) defines the need for arrival times to be within the customers time window. The depot also has a time window associated with it (opening and closing time) which all vehicles must adhere to (2.9).

$$\sum_{i \in N} d_i \sum_{j \in \Delta^+ i} x_{ijk} \leq \sum_{k=0}^{K} k_i \qquad \forall k \in K, \tag{2.10}$$

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ijk} \tag{2.11}$$

Equation (2.10) guarantees that the vehicle not exceed its capacity limit. The objective function is defined by Equation (2.11) which illustrates the desire to minimize the total path cost. Note that the objective is the sum of the path costs per vehicle route. The path cost does not include the service or waiting time. Time is only a constraint that causes some routes to be infeasible.

2.3 Coalition Forming

Within the field of Artificial Intelligence and Robotics exists a field of study called coalition forming. This field deals with the problem of determining the optimal grouping of agents to different targets and how the robots should get there. This field of study is a precursor to the development of the Swarm Routing Problem. In Gerkey [20] a taxonomy for the different types of robot tasking problems is developed. Gerkey

defines the tasking problem and maps it to a combinatorics set covering problem. By doing this the complexity of the problem is defined and he explores to define a variety of solution techniques. The problem in Gerkeys research termed the single-task robots, multi-robot tasks, time-extended assignment problem is the same premise that is proposed in this research. However the problem is not expanded by mapping the problem model from the VRPTW as is done here.

2.4 Path Planning

The process of path planning develops a sequence of steps to get from a starting point to an end point based on the internal representation of the terrain to be traversed. The path is then optimized for a minimal cost. Cost to get to a point may be just the distance traveled but can include fuel used, time, or hazards within the terrain. A path planning algorithm is a method used to calculate a path plan given knowledge of the path environment and a set of conditions or constraints that must be followed. Once generated, the plan is either executed, or further refined to gain improvement. It is generally assumed that once the plan is given, the machine becomes autonomous and can no longer interact with the planner [15].

Any path-planning algorithm must use a representation of the terrain in order to determine what the cost of a path is. Usually this is a grid which associates coordinates with locations in a terrain field. The points of interest that need to be visited are known as vertices or nodes. Connectivity between these vertices is expressed by paths, arcs or edges. From this general statement of the purpose of path planning the concept divides into two realms known as configuration space and trajectory space.

"Configuration space" is a problem definition where all possible positions of some physical system are defined. The solution involves determining the set of actions (torque, translation, etc.) necessary to move the system from an initial state to a goal state, where both of these states exist within the defined set of states and the system never goes outside the set of possible states. Configuration states are most often used

in the development of 3D robotic motion as they result in a set of states the robot must pass through to reach its end goal.

"Trajectory space" also deals with a space of valid and invalid positions within a space of possible configurations. The result, however is not a set of states but a set of trajectories along a path that the physical system passes through from the initial state to the goal state. The solution is determined by first defining a path between the start and end positions that passes through the valid and invalid positions. The path is then incrementally adjusted until a valid or optimum path is formed. These adjustments are made by determining where the line intersects with obstacles and then explores the subsection of the path to determine how it can be altered to avoid the invalid area.

There are many methods for dealing with both of these problems space definitions. For small-scale path planning problems search trees are often used to determine the optimum path the physical system must follow [26]. As complexity increases into 3-dimensions and many hazards, evolutionary methods have been shown to be effective for determining optimal solutions [25].

2.4.0.1 Evolutionary Solutions in Trajectory Space. One of the first examples of evolutionary computation applied to trajectory space is used in the Evolutionary Planner/Navigator (EP/N) created by Xiao and Michalewicz [57]. The planner treated the path from the start to end position as the chromosome. By manipulating the location of intermediate points through the genetic operations of the Evolutionary Algorithms (EA) the path was evolved into an optimal path consisting of a set of way-points. This work was later refined by Sugihara [46] and then modified into a two objective problem by Castillo [6]. These previous papers defined the idea of starting with a straight line path and then evolving it into a path that navigated around obstacles and hazards areas.

In order to deal with the flight dynamic constraints for UAV path planning, Slear used a concept called B-spline curves [44]. This method allowed for the development

of controllable path changes while the path is being evolved. A path is defined by sets of tuples (X_n, Y_n, Z_n) where each set consists of three tuples. By modifying which sets are placed in the final path and by generating random tuple sets via mutation optimal paths are generated. These paths can be formed according to any number of objectives. In the case of Slear these objectives were cost and risk [44]. The cost of the path entails the fuel required to climb or travel the distance of the path. The risk is the proximity to enemy locations or height above ground (higher altitudes increase radar visibility). These objectives usually do not complement each other, as a low cost path generally has a high associated risk and a low-risk path generally involves high cost (through the need for flying low-over uneven terrain).

The domain of the problem consists of an operational $n \times m$ grid space, a terrain grid G (2.12) and a location set L (2.13). The location set is the set of targets from the VRPTW described in Section 2.2.

$$G \in (n-1) \times (m-1) \tag{2.12}$$

$$l_i \in n \times m \forall l \in L \tag{2.13}$$

The problem is constrained by a heading change requirement defined in (2.14). This constraint indicates that changes to a paths heading can not exceed a change greater than 45°. This constraint exists to ensure that solution paths do not require heading changes greater than a UAV is capable of achieving. This constraint is of course dependent on the type of vehicle being modeled.

$$\forall p_0 ... p_n \in P, \Delta\theta(p_i, p_{i+1}) \leq 45^\circ \tag{2.14}$$

where θ is the inbound heading at p_i

The objective of the path planning problem is to develop a minimum cost path P* from all $l_i \in L$ to all $l_j \in L$. A path is defined as a set of points p_n such that $p_n \in P$. The determination of this path cost is found in five parameters divided into two separate objectives making this a multi-objective problem. The path value Φ_{path} is the Euclidian ground distance between two points. The climb value Φ_{climb} is the change in altitude that occurs over some path length. It is found via the summation of the change in altitude from intermediate points in the path being examined. Increases in altitude are to be avoided while decreases in altitude are favorable as they require less power. Detection Φ_{detect} is a value based on if the vehicle is in a known detection area and for how long which can also be thought of as risk. This value can be calculated as either a binary value, which would indicate that detection areas are to be completely avoided, or some normalized value based on how long the vehicle is in the detection area. The kill cost Φ_{kill} is associated with being in range of the enemies ability to fire. This is similar to the detection area but is generally smaller in area. The kill cost can also be thought of as a binary value (i.e. always avoid) or some normalized value (i.e. avoid as much as possible). The "terrain objective" $\Phi_{terrain}$ is a measure of how many points the UAV can be seen from. Minimizing this objective ensures the UAV can not be seen by known and unknown threats. This measure is similar to the altitude cost requirement, however by inherently desiring low altitude flight, as this decreases the odds of being spotted, it serves to effectively contribute to path cost optimization even though it is calculated differently. These five objectives define a multi-objective optimization problem.

2.5 Multi-objective Evolutionary Algorithms

Evolutionary and stochastic search algorithms have been applied to a variety of optimization problems with great success [4, 8, 19, 41]. A review of the concept of evolutionary algorithms is found in Appendix A. Evolutionary algorithms are capable of providing polynomial time solutions for most NP and NP-Complete problems [9] that would otherwise require an exponential or intractable solution time.

NP or "Non-deterministic Polynomial time" combinatorics problems [10] are those problems where finding a solution requires a non-deterministic Turing machine and is intractable otherwise. The solution, once found, can be verified in polynomial time. The most difficult NP problems are termed NP-Hard, which refers to problems that can not be solved through exhaustive search in either tractable or intractable time frames. Some NP-Hard problems can be transformed into NP-Complete problems if the NP-Complete solution also applies to the NP-Hard problem. The idea is that NP-Hard problems constitute the hardest set of problems to solve in a tractable time due to an unlimited solution space while NP-Complete problems comprise a subset of those problems with a limited solution space that is still very large. Figure 2.2 shows the overlapping area these problem types exist in. A classic NP-Complete problem is the TSP. The TSP consists of a set of n points on a grid connected by weighted edges; the objective is to determine the shortest circuit that goes through all n points and ends where it began. For a small number of points this is not difficult, however the solution complexity time of the problem is $\frac{(n-1)!}{2}$ and the number of solutions that must be analyzed quickly increases to an intractable number [51].

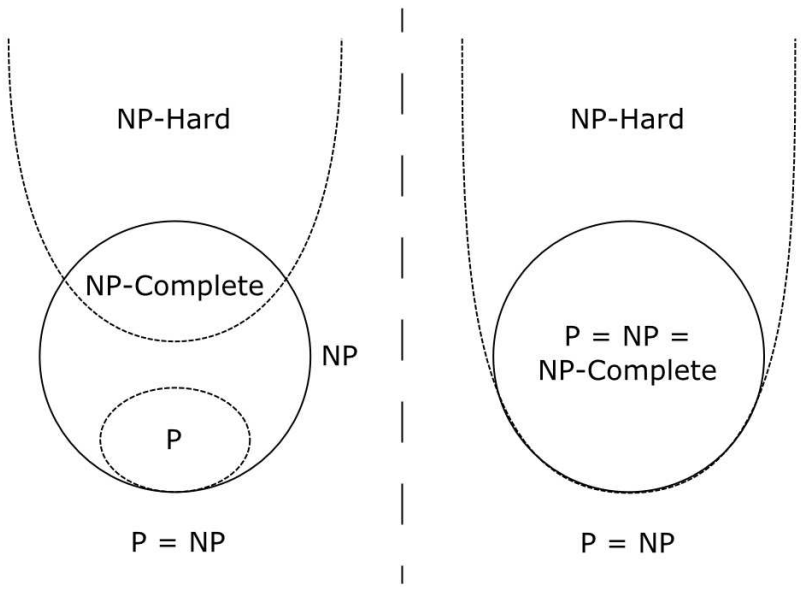

Figure 2.2: NP-Problem subdivisions.

For these many different types of problems a variety of applicable solution techniques exist. Many deterministic solutions exist for small or constrained NP problems which rely on search trees. However there is a limit to how effective deterministic (or even heuristic based) search methods can be, especially as problem size and complexity grows.

One of the interesting aspects of NP problems is that many of them can also be modeled as multi-objective problems. There are two conflicting effects of a multi-objective problem. The first is that the problem is often more useful because it more closely approximates reality, but this comes at a cost of the second effect: increased complexity. The VRP is hardly realistic, however the VRPTW described earlier in this section is closer to reality, and if more constraints were applied, such as heterogenous vehicles, back-hauls (pick up and delivery), or multiple depots, the problem would become even more realistic. While this makes the solution of the problem more valuable it also makes an optimal solution that much more difficult to obtain.

With knowledge of the effective use of evolutionary algorithms and the need for multi-objective problem solutions, recent work has focused on developing multi-objective evolutionary algorithms [3, 33, 62]. MOEAs are basically the same as a standard GA with the difference of how solutions are evaluated and ranked. In a multi-objective solution the concept of dominance exists. A solution is said to dominate if there is no other solution that can improve one of the objectives without simultaneously reducing another [9].

By examining the dominance of different solutions and ranking them accordingly, a more accurate selection of effective solutions can be made for future generations. Also, by ranking across multiple objectives the resulting solution achieves optimal performance across all objectives without being biased toward any one objective. When discussing optimality in a multidimensional space the concept of the pareto front becomes beneficial. The Pareto front is the set of non-dominated, feasible solutions. More recent MOEAs [63] [13] use this understanding of the Pareto front in

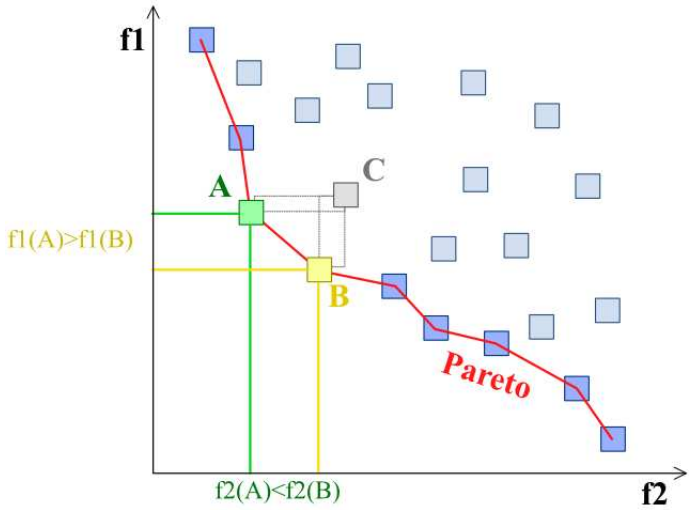

Figure 2.3: Dominance example for two objectives f1 and f2 [56].

order to track which genotypes are developing better solutions. This concept of the Pareto front is visualized in Figure 2.3. Notice that the resultant set of solutions in the front provides solutions with different tradeoff values. Which solution is actually used is a decision made by a user or by some pre-determined rule.

An examination of the literature reveals a growing appreciation for the use of MOEAs in complex problems, such as the VRP [48] [29]. The reason for this is that MOEAs are better able to navigate the highly irregular solution space that exists within the VRPTW. What has also been shown is that multi-objective approaches not only develop good solutions but are also better than biased single objective solutions for the optimization of any of a problems multiple objectives [33] for certain problems.

2.6 Self Organized UAV Swarms

The idea behind self organization is that developing the control structure to make a group of UAVs work together is computationally infeasible. SO theory defines a method for applying simple rules to individual vehicles, which when put into large groups develop the emergent behavior relevant to the mission [7].

Within UAV simulation a control methodology must be established that defines how each UAV flies and makes decisions. While it is possible to manage each UAVs

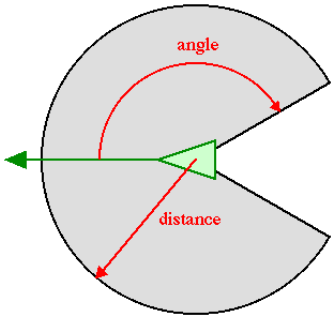

Figure 2.4: UAV local sensor pattern [37].

flight with a deterministic method that selects optimal flight paths this is an extremely computationally intensive operation. For this reason, a method of self-organized control is used. Self-organization [9] refers to an emergent property that exists within complex systems composed of very simple entities. An example of this is the flocking of birds. Each bird has no global knowledge nor understanding of direction, they only know to stay within a certain distance of its neighbors and avoid obstacles. Figure 2.4 illustrates the local sensor area available to the swarm member. The net effect of each bird following these simple rules is the emergent flocking behavior we see in the real world. These rules were defined by Reynolds in his original paper on the subject [38].

It is difficult to predict what low-level rules result in the macroscopic behavior desired in a self-organized entity. For this reason much of the current work in SO focuses on evolving and optimizing existing rule sets that achieve the desired behavior [36]. While tuning the specific parameters is difficult and time consuming, defining the rules is not as complicated. For example, in a swarm of UAVs, the application of Reynold's three original flocking rules plus a rule later defined in [54] for flocking can be used. These rules are illustrated in Figures 2.5-2.8.

The first two items can actually be seen as a single rule to stay within a certain range of nearby flock members. Each one of these rules represents a vector of influence for the swarm member. The direction the swarm member flies is a summation of these four vectors. The result of these four rules is that each vehicle avoid crashing into another, all members fly at the same speed, and the entire flock moves toward some

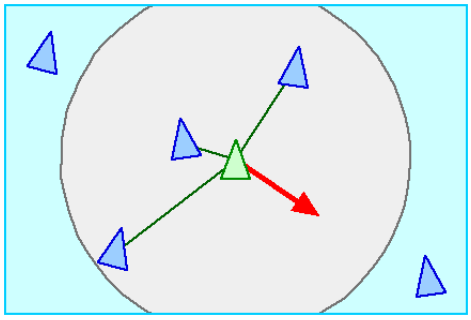

Figure 2.5: Collision avoidance: avoid hitting nearby swarm members [37].

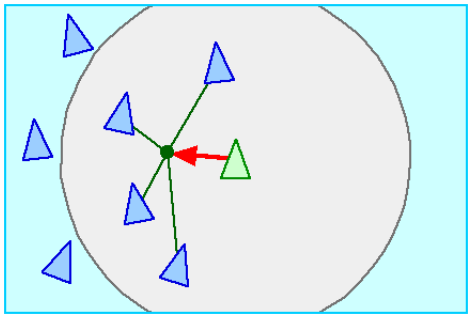

Figure 2.6: Swarm centering: stay within sight of nearby swarm members [37].

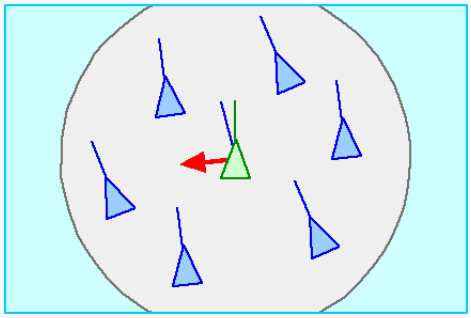

Figure 2.7: Velocity matching: match velocity of nearby swarm members [37].

objective point. The basic idea of self- organization is not that complicated; realistic simulations, unfortunately, are.

There are quite a few simulations of self organized behavior available online [30] though these are generally limited to 2D simulations, with the exception of the Air Force Institute of Technology (AFIT) UAV simulator and a few others [37]. The problem with conducting simulations is the inherent complexity of managing the global information of the swarm and its environment ($O(n^4)$ for the 4 basic rules per

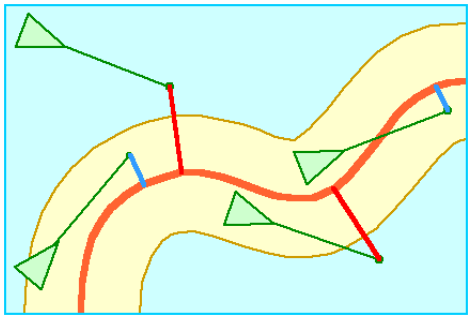

Figure 2.8: Migration: maintain velocity to a set way point [37].

UAV). Recent advances in computer hardware, data structures, and parallel software have enabled more advanced simulations but the problem complexity is still very relevant, especially as more detailed and realistic simulations are required.

2.7 Summary

This chapter presents a short history of previous research that supports this effort. The topics of the VRPTW, path planning, MOEAs, and SO are discussed in their relation to this thesis investigation. Previous research is also explored as part of topic development. The problem domain consists of UAV routing. The solution approaches consist of MOEAs (as a solution to the routing problem). Next, the SRP is defined as an alternate routing model and along with the VRPTW, is formulated as a multi-objective optimization problem.

III. Problem Definition

This chapter formulates the SRP in a manner similar to that of the VRPTW in Chapter II. The SRP is a modification of the VRPTW that allows for a more realistic routing of UAVs to targets. The chapter concludes with a multi-objective formulation of the VRPTW and SRP.

3.0.1 The Swarm Routing Problem. In almost all variations of the VRP, or VRPTW, it is assumed that all vehicles depart from the depot location to different targets and only one vehicle visits each target [51]. This problem model is appropriate because each vehicle is generally assumed to be ground based. However, the use of a UAV swarm introduces an interesting aspect which, up to this point, has not been dealt with. When dealing with a swarm of UAVs and multiple targets to visit, it is desirable to have the ability to route the swarm between targets in the most efficient manner possible. The reason for this is that often many targets exist on a battlefield which need to be visited in a timely manner, while also utilizing resources in the most economical fashion possible. It would be much more efficient to be able to consider UAVs as a dynamic group rather than indivisible units that can only visit one location at a time. This implies an imperative to take advantage of the divisibility of the swarm and route subgroups of UAVs to different targets, as it is deemed efficient to do so, and then regroup at other targets.

By viewing the problem in this manner the value of importance changes from edge costs between targets to the distance traveled by each UAV. Within the VRPTW (and VRPs in general) each vehicle is seen to have some capacity associated with it that is used to satisfy each customer. While this works for ground based delivery routing, it would be more appropriate to view target satisfaction as the number of UAVs on target in some time window. The path cost associated with a single vehicle is then more reflective of the distance that needs to be traveled, and the use of many vehicles capable of being routed through multiple targets (causing splits and joins in the group in the process) presents a more realistic and useful problem model.

The problem construction is modeled off the structure of the VRPTW found in Toth [51] and Tan [47]. The problem domain consists of a network $G = (V, A)$ where $V = \{v_0 v_1, ...v_n\}$ and v_0 is the depot. The set of edges is defined as, $A = \{(v_i, v_j) \in V, i \neq j\}$, where each edge has associated with it some cost $c(v_i, v_j)$. The cost of the edge is the cost of travel from target i to target j. For now we assume path cost as some constant travel speed for each UAV making the travel cost simply the Euclidean distance between points.

A time window exists for all customers where, E, represents the earliest start time and, L, the latest arrival time. The latest arrival time is the point at which the UAV can arrive and still complete the service time defined by S. If the vehicle arrives earlier than E, it incurs a waiting time, W, which is the difference between the arrival time and E. The total time a vehicle takes to complete it's route is the summation of all route path travel costs, waiting times, and service times ($\sum c_{ij} + \sum w_i + \sum s_i$). The total path time must not exceed the latest arrival time (i.e. closing time) of the depot.

Each customer also has associated with it some demand, D. The demand is an indication of the number of UAVs that need to be present at the target within its time window for the required service time. **This is one of the key differences between the SRP and VRPTW.** Instead of demand being satisfied by a UAVs capacity, it is satisfied by the number of UAVs at the target. The service time indicates the amount of time the UAVs are required to be on target.

There exists K homogenous UAVs each of which has some travel capacity, F, and unit deliverable capacity. Groups of UAVs are classified as a swarm. A swarm can split into one or more sub-swarms, join with other sub-swarms into a larger swarm, and travel along path edges together as a swarm. It is assumed that join and split operations only occur at targets in order to simplify problem complexity. The travel capacity is not a deliverable value as it has been in previous versions of this problem, it is only an indication of how far the individual UAV can fly. This constraint can

be viewed as the UAVs power supply limitation. The deliverable value that satisfies the target is equal to the total number of UAVs present in a given location at a given time. This value fulfills the demand requirement of the target during its service time.

The solution to the problem is the same as the VRPTW, a list of ordered targets for each vehicle, such that the visitation to each target fulfills all target needs without violating any time or demand constraints. Note, that the cost of a route is not the total time the route takes to complete, it is only the sum cost of the edges the vehicle traverses. The objective remains the same: determine the set of paths for the UAVs such that the total distance is minimized.

The following is a mathematical programming formulation of the SRP, based on the definition found in Toth [51]. Vehicles are defined within the problem by their inclusion in a flow variable, x_{ijk}, which is a binary value indicating if UAV, k, exists on the path that connects $(i, j) \in V$ at any point in the solution. A time variable, ω_{ik}, indicates the start time of UAV k at location i. The subscript $j \in \Delta^{\pm}(i)$ indicates the set of edges from i to j where j is not equal to i, the plus or minus indicates either a forward or backward move along the path.

A_{ij} - Edge cost between i and j

V_n - Network vertices for n target (v_0 is the depot)

E_n - Earliest arrival time of target n

L_n - Latest arrival time of target n

S_n - Service time of target n

D_n - Demand of target n

K - Set of UAVs

F_k - Travel capacity of UAV k

$$x_{ijk} \in \{0, 1\} \qquad \forall k \in K, (i, j) \in A, \tag{3.1}$$

$$\omega_{ik} \geq 0 \qquad\qquad i \in N, k \in K, \tag{3.2}$$

Equations (2.1) and (2.2) define the flow and time variables used. The flow variable is a binary value that indicates vehicle, k, travels from location, i to j, if equal to one, and zero otherwise. The time variable specifies the start time at location, i, by vehicle k.

$$\sum_{j \in \Delta^+(i)} x_{ijk} = 1 \qquad\qquad \forall i \in N, \forall k \in K, \tag{3.3}$$

$$\sum_{j \in \Delta^+(0)} x_{0jk} = 1 \qquad\qquad \forall k \in K, \tag{3.4}$$

$$\sum_{i \in \Delta^-(n+1)} x_{i,n+1,k} = 1 \qquad\qquad \forall k \in K, \tag{3.5}$$

$$\sum_{i \in \Delta^+(j)} x_{ijk} - \sum_{i \in \Delta^-(j)} x_{ijk} = 0 \qquad \forall k \in K, \forall i \in N, \tag{3.6}$$

Equations (3.3)-(3.6) define the edge constraints of the graph in a solution. They indicate that the each vehicle visits a customer only once (3.3), that all vehicles must start from the depot (3.4), that all edge costs are symmetrical (3.5), and that all vehicles must return to the depot (3.6). Note, in Equation (3.3) the lack of the summation over K which removes the constraint that each customer be visited by only a single vehicle.

$$x_{ijk}(\omega_{ik} + s_i + t_{ij} - \omega_{jk}) \leq 0 \quad \forall k \in K, (i,j) \in A, \tag{3.7}$$

$$e_i \sum_{j \in \Delta^+(i)} x_{ijk} \leq \omega_{ik} \leq l_i \sum_{j \in \Delta^+(i)} x_{ijk} \quad \forall k \in K, \forall i \in N, \tag{3.8}$$

$$e_i \leq \omega_{ik} \leq l_i \qquad \forall k \in K, i \in (0, n+1), \tag{3.9}$$

Equations (3.7)-(3.9) define the time constraints of the problem. Equation (3.7) indicates that the arrival time at location i plus the service time and travel time to the next location must equal the arrival time at the next customer. Equation (3.8) defines the need for arrival times to be within the customers time window. The depot also has a time window associated with it (opening and closing time) which all vehicles must adhere to (3.9).

$$\sum_{i \in N} d_i \sum_{j \in \Delta^+ i} x_{ijk} \geq \sum_{k=0}^{K} k_i \qquad \forall k \in K, \tag{3.10}$$

$$\sum_{j \in \Delta^+ i} c_{ij} x_{ijk} \leq F_k \qquad \forall k \in K, \tag{3.11}$$

Up to this point the formulation is basically been the same as the VRPTW with the exception of Equation (3.3). Equations (3.10)-(3.11) are what separate the SRP from the VRPTW. Equation (3.10) indicates that the demand of each customer is satisfied by the number of vehicles on location and that that number must be either equal to or greater than the demand of the target. Equation (3.11) indicates that the total cost of the path for a vehicle not exceed the vehicles flight cost limit.

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ijk} \tag{3.12}$$

The single objective function is defined by Equation (3.12) which illustrates the desire to minimize the total path cost for all vehicles. The path cost does not include the service or waiting times. Time is only a constraint that causes some routes to be infeasible, the cost is the total distance traveled.

This formulation has introduced the SRP as a modification of the VRPTW. By changing the constraints of customer visitation and how a customers demand is satisfied the problem becomes a more realistic model for routing UAVs to multiple targets within a time window. Up to this point problem and the VRPTW defined in Chapter II have only been single objective formulations. In the next section the problems are reviewed in terms of multiple objectives.

3.1 Multi-objective Formulation for VRPTW and SRP

In Section 2.2 the VRPTW is defined and in the previous section a variant, the SRP, is defined. The objective functions for these two problems indicates that only path length is of critical interest. Even though path length is a primary objective it is not the only objective that can be optimized for in the solution. Consider the following situations that may occur within a problem:

- **Vehicle exceeds its capacity to serve a route** - when this happens the route can be split into 2 or more routes. The split is made when the customer demand causes the capacity of the vehicle to be exceeded. If the original route (1,2,3,4,5,6) causes the vehicle capacity to be exceeded at customer 4, the route is split into two routes (1,2,3) and (4,5,6) and a new vehicle is added to the solution. This would be repeated in all routes until the capacity is met.
- **Vehicle arrives early to a customer** - when this happens the service time for the customer is increased by the time spent waiting
- **Vehicle violates a time window by arriving late** - a new vehicle and route are added, splitting the route as described when capacity constraints are encountered.

From these situations we see that the solution to alleviating capacity and time violations is to increase the number of vehicles (and routes). Increasing the number of vehicles is, of course, regressive to the development of optimal paths lengths. This is due to the introduction of depot travel times for each new route. Every time a new

route is added the vehicle must first travel from the depot to a location, making the addition of new vehicle routes generally cause an increase in total path length (though not always). It is therefore, advantageous to define 3 objectives to minimize for: path length, vehicle count and total wait time. By optimizing for these three objectives we seek solutions with complementary aspects, and better solutions overall. The objective functions now consist of the following equations:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ijk} \tag{3.13}$$

$$\min \sum_{k=0}^{K} k \tag{3.14}$$

$$\min \sum_{k \in K} \sum_{(i,j) \in A} t_{ik} x_{ijk} \tag{3.15}$$

Equation (3.13) defines the minimization of the path length. Equation (3.14) defines the minimization of the number of vehicles used. Equation (3.15) defines the minimization of the wait time where the variable t is defined in equation (3.16) for the VRPTW and (3.17) for the SRP. The wait time for the SRP is defined as the difference between a vehicles arrival time and the arrival time of the latest vehicle, if the latest arriving vehicles time is past the earliest arrival time of the customer. The latest arriving time is used because operation on a customer can not begin until all vehicles are present. These objective functions apply to both the VRPTW and SRP.

$$t_{ik} = \left\{ \begin{array}{l} e_i - w_{ik}, \text{ if } E > w_{ik} \\ 0, \text{ otherwise} \end{array} \right\} \tag{3.16}$$

$$t_{ik} = \left\{ \begin{array}{l} e_i - w_{ik}, \text{ if } E > w_{ik} \cap E > w_{iu} \\ w_{iu} - w_{ik}, \text{ if } w_{iu} > E \\ 0, \text{ otherwise} \\ \text{where u is the latest arriving vehicle} \end{array} \right\} \tag{3.17}$$

3.2 Purpose in multi-objective formulation

Optimizing across multiples objectives is done with two purposes in mind. First, to support the idea that a multi-objective formulation is capable of navigating the solution space more effectively than optimizing for only a single objective. Since the objectives complement each other it would seem logical that optimizing over all of them would achieve better results. What is also noteworthy is that by optimizing for these different objectives, solutions with decreased path lengths should be found as opposed to optimizing solely for path length. This idea was first proposed and tested in Ombuki [33] with beneficial results (i.e. benchmark values were not made worse from the multi-objective approach compare to the single objective approach).

The reason multi-objective formulation is more effective is because the problem under consideration has such an irregular solution space. Time constraints introduce irregularities to the Pareto front such that non-dominated solutions become more isolated. This problem is exacerbated as more constraints are applied to the problem such as heterogenous vehicle fleets or pick-up and delivery problems. Only by optimizing across multiple objectives can the solution space be traversed accurately enough to allow the determination of non-dominated solutions. We now proceed with the high level solution design to this fully formed multi-objective problem.

3.3 Summary

This chapter introduced the SRP routing problem variation as a model for routing swarms of UAVs. By eliminating the restriction of single city visitation for each vehicle and changing target satisfaction to be based on vehicle count the SRP

serves as a better problem model for UAV routing. A multi-objective formulation for the both the VRPTW and SRP shows the objectives can be optimized for: path length, vehicle count, and wait time per vehicle. The next chapter introduces the high level solution design for these two problems.

IV. High Level Concept Design

This chapter presents the high level design that forms a solution to the multi-objective VRPTW and SRP defined in Chapter III. An analysis of problem complexity precedes a development of the algorithmic solution. This development follows a process of determining the type of solution, the structure of the algorithm itself, and the individual components that make up its design.

4.1 Design Objectives

The design objectives for this research are to develop a solution procedure for the multi-objective routing problems described in the previous chapter. This solution must return information that can be integrated with previous work on UAV path planning [44] and simulation [11]. The end result of this design is a fully developed form of an algorithm to be applied to the VRPTW and SRP whose output is a set of feasible vehicle routes.

A discussion of the problem complexity is performed in order to illustrate design requirements of the algorithm. An introduction to the form of the required solution is then shown; this form requires two aspects: a selection method and set of genetic operators. A description of two multi-objective EAs is given in support of the first requirement. This is followed by a set of genetic operators that compose the GA solution for the VRPTW and SRP, each contained in its own respective section [1].

4.2 Problem Complexity

The VRPTW is NP-Hard [10, 51] meaning that no polynomial time solution exists unless $\mathcal{P} = \mathcal{NP}$. This complexity valuation is determined by analysis of the VRP as NP-Hard [45] and through restriction, determining that the VRPTW is also

[1]The term evolutionary algorithm and genetic algorithm are almost interchangeable and often used as such. Evolutionary algorithm is usually considered a broader term encompassing many different type of evolutionary computation. Genetic algorithms refer exclusively to optimization and search algorithms which employ chromosomal solution manipulation, where the chromosome is directly related to the solution (this excludes genetic programming)

NP-Hard. For a fixed number of vehicles the problem becomes NP-Complete [51] for either problem variation. The SRP, as an extension of the VRPTW, can likewise be classified as NP-Hard in the general sense and NP-Complete for fixed fleet sizes.

To determine the solution space complexity we start with the complexity of a restrained problem, called the TSP. In TSP the number of vehicles is constrained to one, making the size of the problem $1/2(n-1)!$ for $n > 2$, where n is the number of customers/targets in the problem. The solution space size of the VRP, S, is approximated by Equation (4.1).

$$S \approxeq \frac{\exp^{(\pi\sqrt{2n_!/3})}(n-1)!}{8n\sqrt{3}} \tag{4.1}$$

This equation is found by combining an integer partition distribution generating function and the the complexity of a single n customer routing problem [55]. The result of this approximation yields a solution space complexity of $\mathcal{O}(\exp^n n!)$. For the VRPTW the solution space is the same as it contains the same number of total solutions as the VRP minus some constant number of solutions made invalid by time constraints. This same idea would apply to any VRP variation (i.e. the SRP solution space complexity is that of the VRPTW). This rapid increase in the solution space is the source for the VRP being such a difficult problem to solve in the general sense.

The complexity of the VRPTW leads to the conclusion that a polynomial time solution most likely does not exist. The best course of action is to then pursue a heuristic based solution most likely structured within a stochastic algorithm [9]. This assumption is backed by published research on the success of stochastic solutions to the VRP and VRPTW [21] [34] [47] [3].

4.3 Multi-objective Algorithm Design Choices

Many applicable algorithm choices exist for solving the VRPTW and most are capable of returning optimal solutions in a reasonable time as seen by the choices

examined in this section. The choices have been constrained to stochastic based algorithms as our previous complexity analysis has shown that a deterministic search process would be intractable.

4.3.1 Ant Colony Optimization. Ant colony optimization is defined by a set of artificial ants, or agents, navigating through a defined search space attempting to optimize some problem solution. The route the ants construct within the search space forms the solution. Each ant exists at some point in the solution space and makes a local decision of where to go next. This decision is inherently stochastic but is based on the idea of the pheromone matrix. In nature, ants lay pheromone as they wander through a space. These pheromones attract other ants to follow the same path. As the ants encounter desirable factors more ants start to follow the same path reenforcing the path. This idea is presented visually in Figure 4.1. While this does not lead to an optimum solution (true best solution) it does often lead to optimal solutions (high value that may or may not be the true best value), especially in Euclidian spaces (specific to routing problems).

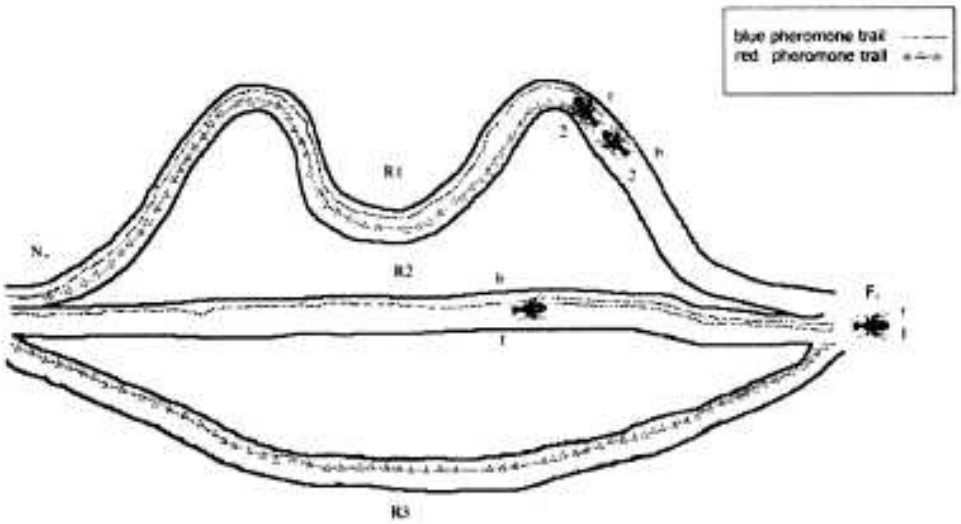

Figure 4.1: ACO Paths Being Created.

The decision of which direction an ant travels, within the ACO model, is defined by the probability definition in Equation (4.2). The equation is the pheromone inten-

sity, τ, times the distance between two points, divided by the summation of all other options calculated the same way. η is a visibility term that denotes the inverse of the distance between two points. This causes the ant to have a preference for shorter routes. This probability is a function of the pheromone matrix which is updated as solutions are found. Altering the update method, seen in Equation (4.4), to update under certain conditions or only for the highest performing ant also impacts the performance of the algorithm. The pheromone matrix is also altered by Equation (4.3) where ρ is the evaporation rate $[0 - 1]$. The net result of these methods is to enforce the use of paths that have lead to good solutions. Algorithm 1 incorporates these ideas into a pseudo code representation.

Algorithm 1 ACO Algorithm

```
1: procedure ACO([best] = max_it,N,τ0)
2:     initialize τij // usually initialized with the same τ0
3:     initialize best
4:     place each ant k on a randomly selected edge
5:     t ← 1
6:     while t < max_it do
7:         for i = 1 to N do
8:             Build a solution by applying a probabilistic
9:             transition rule (e − 1) times.
10:            The rule is a function of τ and η
11:            e is the number of edges on the graph G
12:        end for
13:        Evaluate each solution
14:        if an improved solution is found then
15:            update the best solution found
16:        end if
17:        Update pheromone trails
18:        t ← t + 1
19:    end while
20: end procedure
```

$$\Pr(i,j) = \frac{\tau(i,j).[\eta(i,j)]^{\beta}}{\sum_{allowed_j} \tau(i,j)^{\alpha}.[\eta(i,j)]^{\beta}} \tag{4.2}$$

$$\eta(i,j) = 1/d(i,j), \beta = 2$$

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \Delta\tau_{ij} \tag{4.3}$$

$$S_{upd} \leftarrow \arg \max_{s \in S_{iter}} F(s) \tag{4.4}$$

This is an example of a popular elitist update strategy where for every iteration in the number of generated solutions the one with the highest fitness value is used to update the matrix.

4.3.1.1 Applying ACO to the MOP VRPTW. ACO can be applied to multi-objective problems in one of two ways. A set of different colonies can be run simultaneously attempting to optimize each objective before forming a final solution [19], or a single colony can be run where the probability equation described previously (Equation 4.2) is based not only on the pheromone matrix but also on objective weights that become associated with the pheromone matrix [3]. Each option has advantages and disadvantages. Attempting to solve for all objectives simultaneously has the risk of creating bias toward some objectives, and using different colonies has a higher computational complexity.

Optimizing for all objectives simultaneously requires each of the objectives described in Section 3.1 to be incorporated into the fitness evaluation used to calculate the quality of a solution. Baran suggests a method where each ant initially creates a feasible solution by adding cities to routes as they are deemed feasible (within the time window and vehicle capacity) [3]. From this point, a variation of the pheromone matrix is used to incorporate the value of pheromone paths to the different objectives

being optimized (4.5). This new equation introduces the η_J value which indicates time between cities. In this way, a single colony can be used to develop a solution that is optimal for all objectives.

$$\Pr(i,j) = \frac{\tau(i,j).[\eta_L(i,j)]^{\beta}.[\eta_J(i,j)]^{\beta}}{\sum_{allowed_j} \tau^{\alpha}(i,j).[\eta_L(i,j)]^{\beta}.[\eta_J(i,j)]^{\beta}} \tag{4.5}$$
$$\eta_L(i,j) = 1/d(i,j), \eta_J(i,j) = 1/t(i,j), \beta = 2$$

4.3.2 Particle Swarm Optimization. Particle Swarms are multi agent systems inspired by biological systems. They are a progression in the area of natural computing borrowing ideas from genetic algorithms, ant colonies, and self organization. The general algorithm consists of a population of agents in D-dimensional, real valued space ($\mathfrak{R}^D$). These agents move through the space attempting to locate optimal solutions (as seen in Figure 4.2). Where a given agents movement is based on its own experience, its current location, and the experience of the agents around it.

The agents in the swarm are defined as $X_i = (x_{i1}, x_{i2}, x_{i1}, ..., x_{iD})$, where X is a position in the space. Each agent also has an associated velocity $V_i = (v_{i1}, v_{i2}, v_{i1}, ..., v_{iD})$. The velocity a particle travels is defined by Equation (4.6). The variables c_i represent positive constants while φ represents a uniform random number generation function.

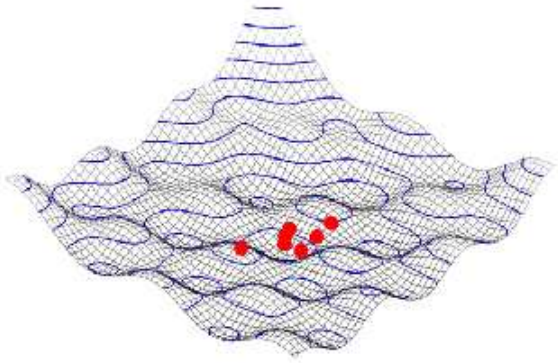

Figure 4.2: Particles on a Solution Space.

The variable w represents the inertia of the problem which corresponds to the experience the agent has had in the search space. The p term is the best value found so far and the best found by a neighbor (p_{gd}). These terms work to move the particles velocity toward better solutions. These best values can be set to either a local best or a global best. The impact of either depends on the problem being solved, though global knowledge is usually more beneficial [58].

$$\begin{aligned} v_{id}(t+1) =& w \times v_{id}(t) + \\ & c_1 \times \varphi_1 \times [p_{id}(t) - x_{id(t)}(t)] + \\ & c2 \times \varphi_2 \times [p_{gd}(t) - x_{id(t)}(t)] \end{aligned} \tag{4.6}$$

Each particle also has a max and min velocity that it can achieve. This value is set by the user and keeps the particle from either shutting down or accelerating out of the solution space. These ideas are incorporated into pseudo code in Algorithm 2.

4.3.2.1 Applying PSO to the MOP VRPTW. In Zhu [61] and Yanwei [58], a Particle Swarm Optimization (PSO) algorithm is applied to a VRPTW. A solution is structured as a $2 \times n + k - 1$ particle, where n is the number of customers visited by k vehicles. Thus a solution to a six dimension problem (with zero as the depot) would be:

$$Node: 00123456$$

$$Index: 26874315$$

The objectives to be minimized are the total path length, the number of vehicles used, and the time lost waiting for customers. In Zhu [61], infeasible solutions (those that violate capacity or time constraints) are penalized instead of being repaired or thrown out. Thus, each objective constraint has some weight associated with it.

Algorithm 2 PSO Algorithm from [7]

```
1: procedure [X] = PSO(max_it,AC_1,AC_2, v_max,v_min)
2:     initialize X // usually x_i, ∀i, is initialized at random
3:     initialize △ x_i // at random, △ x_i ∈ [v_min,v_max]
4:     t ← i
5:     while t < max_it do
6:         for i = 1 to N do
7:             if g(x_i) < g(p_i) then
8:                 p_i = x_i
9:             end if
10:            g = i // arbitrary
11:            // for all neighbors
12:            for j = indexes of neighbors do
13:                if g(p_j) > g(p_g) then
14:                    g = j // index of best neighbor
15:                end if
16:            end for
17:            △ x_i ←△ x_i + φ ⊗ (p_i − x_i) + φ ⊗ (p_g − x_i)
18:            △ x_i ∈ [v_min, v_max]
19:            x_i ← x_i+ △ x_i
20:        end for
21:        t ← t + 1
22:    end while
23: end procedure
```

These values are incorporated into the fitness function that determines how good a solution is and ultimately where a particle goes.

Methods for incorporating multi-objective capabilities include running multiple populations for individual objectives and then combining the best results for a final search [3] and using a memory system to store non-dominated solutions [16]. In Hu [22], a method of randomly selecting non-dominated points to be incorporated into the population is used to ensure exploration of the search space as well as pursuit of all objectives. The implementation of such an algorithm would closely resemble the general implementation in Section 4.3.2. Since PSO algorithms do not rely on structure like GAs the only aspect that is problem specific is the solution structure and the fitness function. The search of the solution space is carried out by a vector definition in Equation (4.6), defined by fitness values, not the solution structure.

4.3.3 Evolutionary Computation. For a review of basic evolutionary computation concepts please refer to Appendix A. Evolutionary computation works on the idea that existing problem solutions can be modified in an intelligent manner in order to allow for the creation of new and better solutions. Evolutionary Computation (EC) methods are very applicable to routing problems because routing solutions can be codified in very simplistic chromosome structures. This simplification allows for easier construction of the algorithm and a faster run time. The next section presents an expanded description of the multi-objective GA.

4.4 Multi-objective Genetic Algorithm Design

Multi-objective GAs differ from single objective GAs in how solutions are stratified. In a single objective algorithm, determining the quality of a solution is a simple matter. If the objective is minimization it is a simple compare operation to the lowest value. For multi-objective optimization this process is not as simple. While it is possible to weight objectives in order to obtain a single value associated with the solution this is not an advisable pursuit. Weighted objectives introduce bias because no weighting procedure can accurately treat the different objectives in a manner such that all objectives are optimized effectively [9]. To accurately classify solutions over a multi-objective domain ,a ranking procedure must be used that takes into account not only the value of the solution across the different objectives but also its proximity to other solutions, which is an indication of the value of the information the solution contains. Chapter II expands on this idea of pareto front searching.

There are three major components to the EA design: the replacement method, the chromosome structure, and the genetic operators. The replacement method determines which solutions are kept after a new generation is created. Within this step the solutions are ranked and discarded. The selection methods chosen are shown in the context of the complete EA. The sections following the replacement method expand on the chromosome structure, population initiation, and genetic operators. The

chromosome structure is a critical step which drives the effectiveness of the entire algorithm and how the different genetic operations function.

4.5 Replacement Method

In this section two replacement methods are shown in the context of the GA they form. Both of these methods rely on non-dominated sorting and objective space distance to rank and select solutions for the next generation. How Pareto ranking and dominance is utilized multi-objective search is discussed in Section 2.5. As arguments can be made for any given selection method for any given problem two algorithms were selected so that their results could be statistically compared. While many different MOEA selection methods exist, Non-dominating Sorting Genetic Algorithm 2 (NSGA2) and Strength Pareto Evolutionary Algorithm 2 (SPEA2) were chosen for their general acceptance within the research community and previous experiments showing their effectiveness [35] [62].

4.5.1 Non-dominating Sorting Genetic Algorithm II. NSGA2 uses an elitist sorting mechanism of the non-dominated points to first organize the solution set [14]. The result of this sorting is a set of solution ranks. The first rank is the hard non-dominated set. Hard non-dominated refers to a point that is not dominated by any other solution, as apposed to soft non-dominated solutions which are only dominated by those points in the first rank. Each decreasing rank is dominated by more points. These points are then compared to each other in order to determine the distribution of points in the current Pareto front and which points contribute best to an exploration of the solution space. This process is called crowding-distance-assignment and is shown in Algorithm 3.

The notation $\Gamma[i].m$ refers to the m-th objective value for the the i-th solution. By using the crowding distance and ranking procedure the solutions are ordered by how "good" they are. A visualization of this idea is shown in Figure 4.3. The next

Algorithm 3 Crowding Distance Assignment

```
1: procedure CrowdDisAssign(Γ)
2:     l = |Γ|
3:     for each i, Γ[i]_distance = 0 do
4:         for each objective m do
5:             Γ = Sort(Γ, m)
6:             Γ[1]_distance = Γ[l]_distance = ∞
7:             for i − 2to(l − 1) do
8:                 Γ[i]_distance = Γ[i + 1]_distance + (Γ[i + 1].m − Γ[i − 1].m)
9:             end for
10:        end for
11:    end for
12: end procedure
```

generation is then filled with the best solution down until the population limit is reached.

The complexity of SPEA2 is $\mathcal{O}(MN^2)$ due to the sorting phase of the algorithm. This complexity more than improves on the factorial complexity of the problem. The only caveat to this is how the genetic operators perform as, based on their complexity, they could dominate the algorithm if not constructed competently. The algorithm is described via pseudo code in Algorithm 4.

4.5.2 Strength Pareto Evolutionary Algorithm II. The SPEA2 was developed by Zitzler [64] as an improvement to the original SPEA algorithm [65]. SPEA2

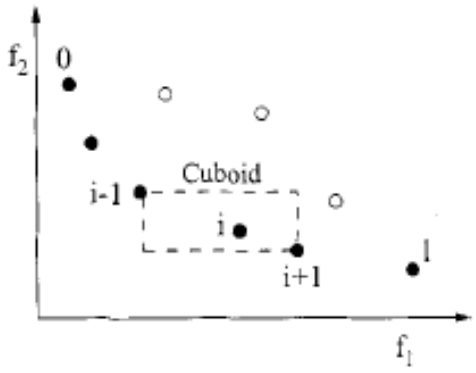

Figure 4.3: Crowding distance calculation. Dark points are non-dominated solutions. [14]

Algorithm 4 NSGA2

procedure NSGA2
  $t := 0$;
  initialize $P(0) := \{\vec{a}_1(0), ..., \vec{a}_\mu(0)\} \in I^\mu$;
  evaluate $P(0) : \{\Phi(\vec{a}_1(0)), ..., \Phi(\vec{a}_\mu(0))\}$;
  while $(\iota(P(t) \neq true)$ **do**
    $R(t) = P(t) \cup Q(t)$
    $\mathcal{F} =$ Fast Nondominated Sort(R_t);
      All non-dominated fronts of R_t
    while $|P'(t)| < N$ **do**
      Crowding Distance Assignment$(\mathcal{F}_i)$
      $P'(t) = P'(t) \cup \mathcal{F}_i$
    end while
    $Sort(P'(t), \geq_n)$
    $P'(t) = P'(t)[0 : N]$; *first N elements in P'(t)*
    crossover: $\vec{a}'_k(t) := r_{s_c,p_c}(P(t))$;
    mutate: $\vec{a}''_k(t) := m_{s_m,p_m}(P'(t))$;
    evaluate: $P''(t) := (\vec{a}''_1(t)), ..., (\vec{a}''_\mu(t)) :$
      $\Phi(\vec{a}''_1(t)), ..., \Phi(\vec{a}''_\mu(t))$;
    select: $P(t+1) = s(P''(t) \bigcup Q)$;
    $t := t + 1$;
  end while
end procedure

uses a strength ranking procedure to stratify solutions. Each solution is assigned a strength value based on the number of solutions that dominate it. First, it is determined how many points dominate each solution, this is referred to as the fitness value. Then each dominated point is assigned the sum of all the fitness values that dominate it, this value is then called the strength value of the solution. It is this strength value that is used to rank the solution. The strength ranking procedure ensures that while good solutions are kept, solutions that are more isolated (but still dominated) are also kept in order to ensure better exploration of the solution space.

After all the solutions are ranked, an environmental selection method reduces the population to a user specified value. Zitzler refers to this value as the archive, which is a misleading term. The archive does not actually save any information from one generation to another. Its purpose is to give the user the ability to control how many points should be saved each generation. By contrast, in NSGA2 once the

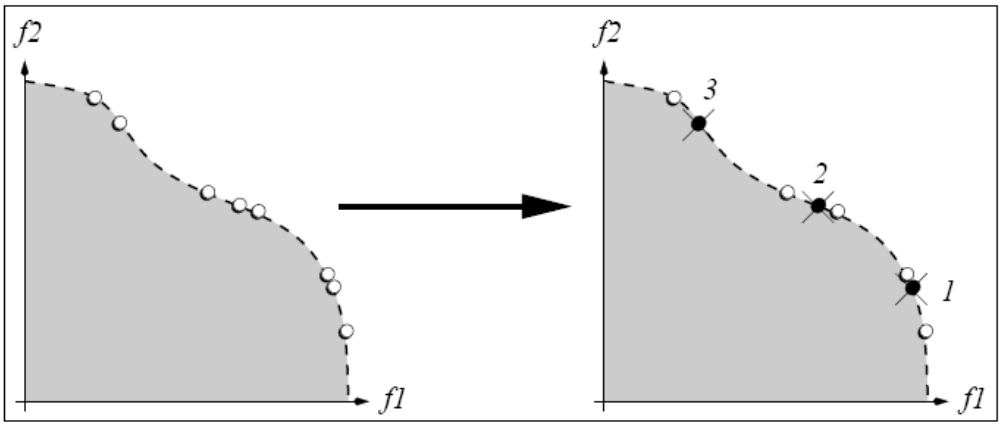

Figure 4.4: Truncation procedure removing non-dominated points [62].

solutions are ranked the crowding distance method runs until the next population is filled. In SPEA2, the user specifies an archive size that indicates how many points should be selected from the ranking procedure to be put back into the population. The intended effect of this is to reduce the number of, what is hoped to be, redundant points passing from one generation to another. Figure 4.4 illustrates this idea by showing how non-dominated points would be removed from a Pareto front.

The run time of the algorithm is dominated largely by the truncation operation. The fitness assignment procedure requires $\mathcal{O}(N^2)$ while the truncation operation is $\mathcal{O}(N^3)$ where N is the number of individuals. While this is a high level of complexity it is still preferable to the factorial complexity described in Section 4.2. Algorithm 5 illustrates the full structure of SPEA2.

4.6 VRPTW Chromosome Structure

Any chromosome solution used in a VRP must be able to specify how many vehicles are required and which cities must be visited in what order. The solution chromosome defines a genotype, which is a code corresponding to a phenotype which is the actual solution. In terms of total information the genotype does not need to contain redundant or implied information. For example, in the VRP it is implied that a route starts at the depot and ends there. Encoding this information in a

Algorithm 5 SPEA2

procedure SPEA2
  $N := populationsize$;
  $\overline{N} := archivesize$;
  $t := 0$;
  generate $P(0) := \{\vec{a}_1(0), ..., \vec{a}_\mu(0)\} \in I^\mu$;
  generate $\overline{P}(0) := \emptyset$;
  evaluateFitness $P(t)$;
  evaluateFitness $\overline{P}(t)$;
  while $(\iota(P(t) \neq true)$ **do**
    envSelection: $\overline{P}(t+1) = nonDomPt(P(t), \overline{P}(t))$;
    mateSelection: $\overline{P}(t+1) = Tournament(\overline{P}(t+1))$;
    recombine: $\overline{P'}(t+1) := r\Theta_r(P(t))$;
    mutate: $\overline{P''}(t+1) := m\Upsilon_m(P(t))$;
    evaluate: $\overline{P}(t+1) : \Phi(\vec{a}''_1(t)), ..., \Phi(\vec{a}''_\mu(t))$;
    $P(t+1) = \overline{P}(t))$;
    $t := t + 1$;
  end while
end procedure

chromosome would therefore be a waste of space. There are three ways to accomplish this, others could be formulated but these have been deemed effective through their repeated usage [39].

A possible solution structure is a bit string where every bit corresponds to an edge [2] in the solution (and every bit is either one or zero indicating whether it is or is not in the solution). This structure is very simple but grows large very quickly and the organization requirement of the VRP lends itself more toward real valued structures anyway. The second structure is a single array of real values representing each target, the order of which indicates the order of visitation. Each route is separated by zeros as shown in Figure 4.5. This structure is more efficient but still requires the use of separators to indicate where a route begins and ends.

In [49] a structure for a VRP chromosome is defined that uses a similar idea as the array structure but attaches each route to a support structure, like that seen in Figure 4.6. The most beneficial aspect of this structure is that changes made to a given

[2] An edge being a connecting line between two points in a graph.

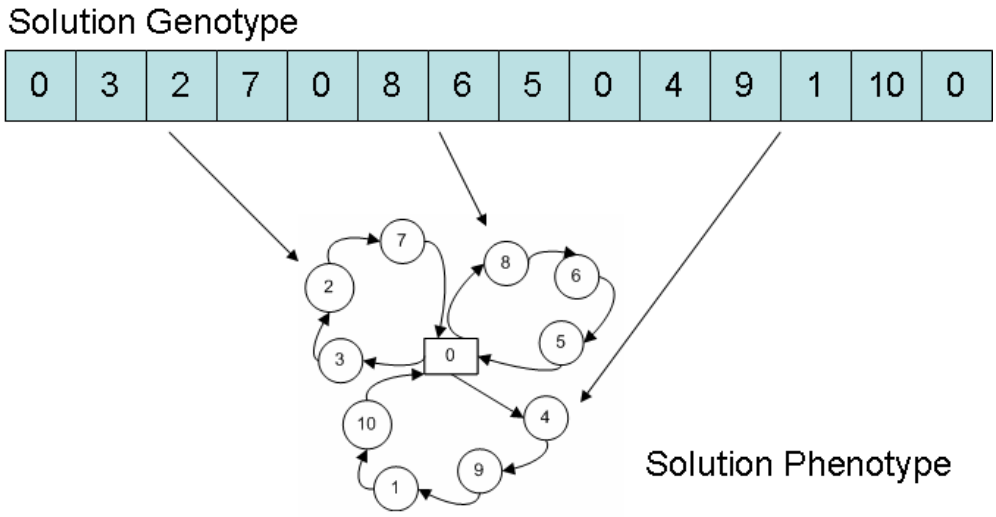

Figure 4.5: A possible chromosome structure for a VRP.

route do not require a shift to the entire array of values. In Tavares [49], this structure is proposed, and shown to be, an affective structure especially for the VRPTW. This structure is also used in previous research by Slear [44] and Russell [40].

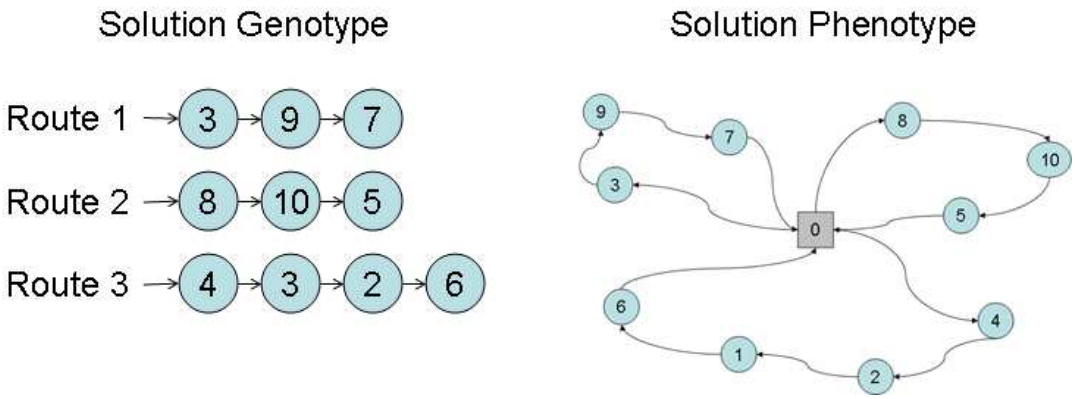

Figure 4.6: GVR chromosome structure for the VRP.

The GVR structure offers many attributes that make it desirable as a chromosome structure. Its information content does not contain redundancies. Each route implies the existence of a departure and return to the depot even though it is not explicitly stated. This is made possible by the support structure that contains and separates each route. It is also desirable that infeasible solutions are not turned into feasible solutions by adding customers but instead only by rearranging and removing

customers. The impact of this is that whenever a solution is checked for feasibility after the addition of a customer it can be safely discarded if infeasible, knowing the solution is a dead end.

4.7 SRP Chromosome Structure

The chromosome structure used for the SRP is essentially the same as for the VRPTW. It consists of single route definitions arranged in a support structure. The only difference is the arrangement of data within the structure. Since each customer must be visited by more than one vehicle at a time the SRP structure must also reflect this. A diagram of this is shown in Figure 4.7.

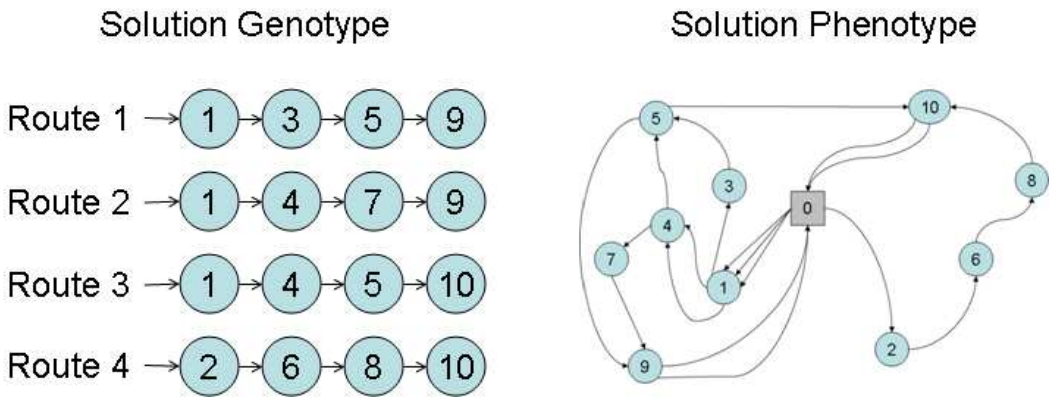

Figure 4.7: Modified GVR chromosome structure for the SRP.

4.8 VRPTW Genetic Operator Development

The operators described in this section are taken from several publications [33, 41,50]. The genetic operator alters a solution in a random manner by either randomly changing the solution or optimizing some sub-section of the solution. This opens two avenues to pursue, simple operators applied many times or the use complex heuristics used to intelligently optimize part of a solution. Classic GAs made use of random operators, however more and more hybrid algorithms incorporate heuristics and local search techniques to great effect [47]. For this research a random crossover method, three random mutations, and a heuristic based mutation operator are used.

4.8.1 Random Crossover. Crossover is the genetic operation that occurs most frequently and ensures that children created from the process are feasible. The operation takes two parents, a donor and a receiver. A random selection of customers is selected from the donor and placed into the first available route in a copy of the receiver, after the customers in the incoming sub-route have been removed. The first available route is the route that when receiving the sub-route does not violate any constraints. If no route exists then the sub-route is added as a new route after the copy customers are deleted in the receiver. The net result of this is a child that is a copy of the receiver but contains some sub route section from the donor. This process is illustrated in Figure 4.8.

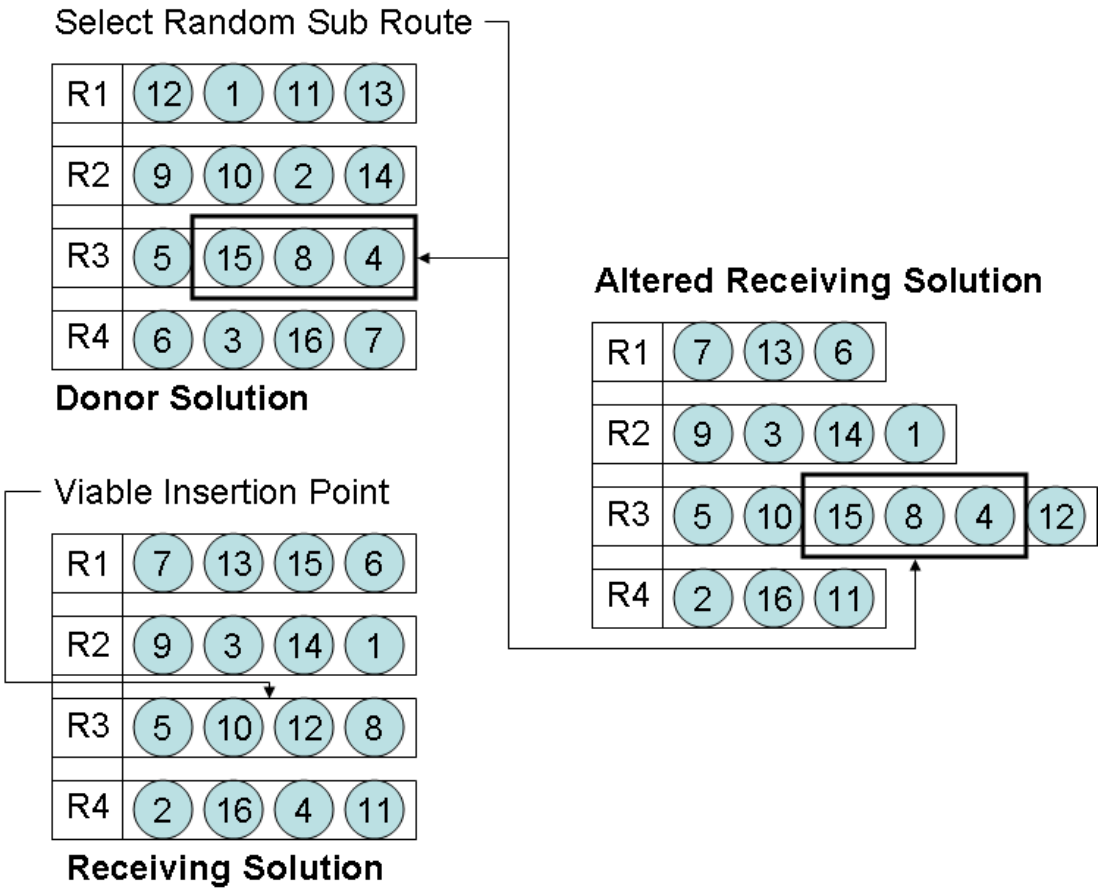

Figure 4.8: Random crossover operator for the VRPTW.

4.8.2 Random Swap Mutation. In swap mutation two random customers in a solution are swapped if doing so does not violate constraints. If constraints become violated the mutation does not proceed. This process is illustrated in Figure 4.9.

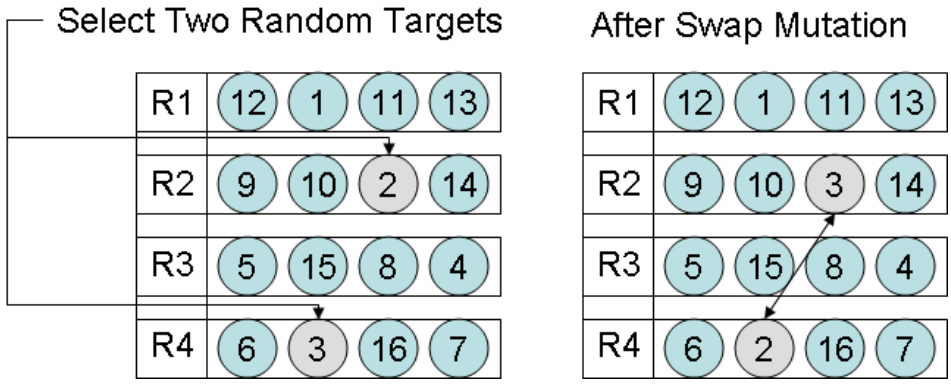

Figure 4.9: Random swap operator for the VRPTW.

4.8.3 Random Inversion Mutation. Select a random sub-route within a solution and reverse the order of visitation. The mutation does not proceed if this results in an invalid solution. This process is illustrated in Figure 4.10. The resultant route may or may not be longer than the original route.

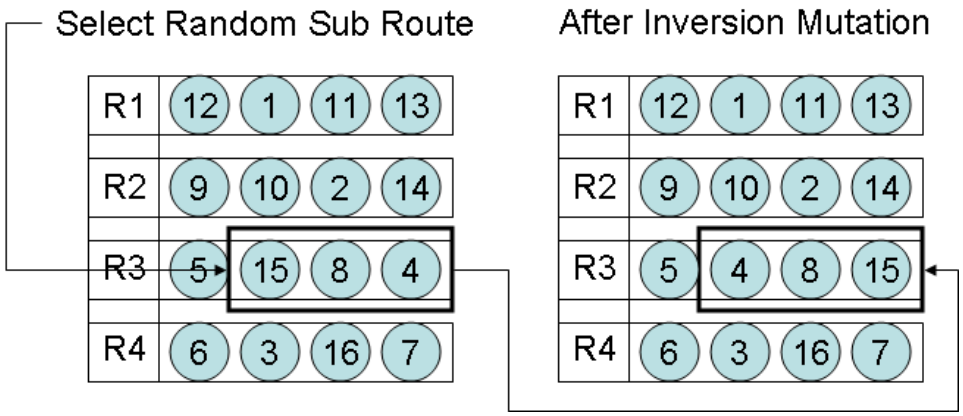

Figure 4.10: Random inversion operator for the VRPTW.

4.8.4 Random Insertion Mutation. Move a random customer to a random location in the solution while ensuring feasibility. It is possible to create a new route

with probability $\frac{1}{2V}$ where V is the number of vehicles [50]. This process is illustrated in Figure 4.11.

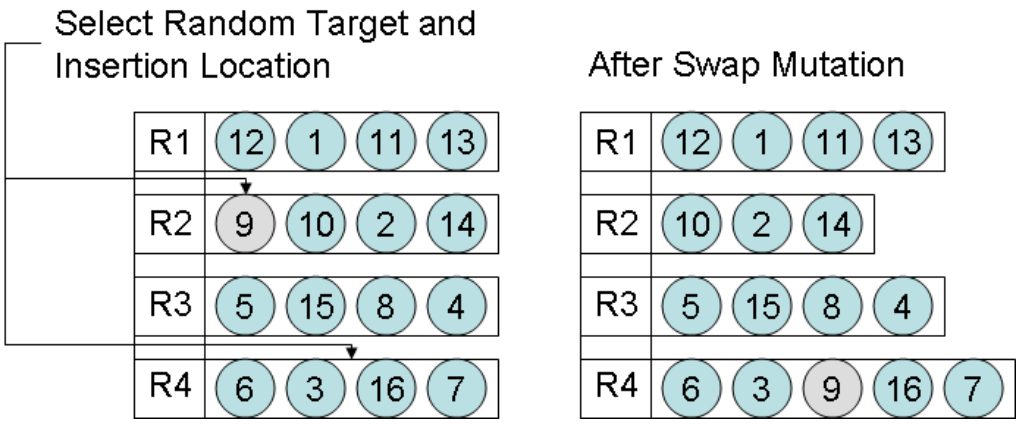

Figure 4.11: Random insertion operator for the VRPTW.

4.8.5 Best Route Cost Mutation. This operator randomly selects a route within a solution and optimizes its construction by rearranging customers within that route. This is accomplished by first searching the route and determining which of the customers is closest to the depot. This customer then becomes the first customer. The closest customer to this customer that is in the route is then moved next to the the first customer and so forth, creating a route based on customer proximity. Customers that can not be feasibly added are moved to a new route. It is always assumed that a single customer within a route is valid, without this assumption the problem would not be solvable. The construction of a new route proceeds in the same way, placing customers by order of proximity. This process is illustrated in Figure 4.12.

4.9 SRP Evolutionary Operator Development

The SRP genetic operators are variants of the VRPTW operators altered to take into account the different structure of the SRP solutions. The difficultly in developing these operators is ensuring the validity of the child genotype. Since the chromosome contains location sensitive information across two dimensions, as opposed

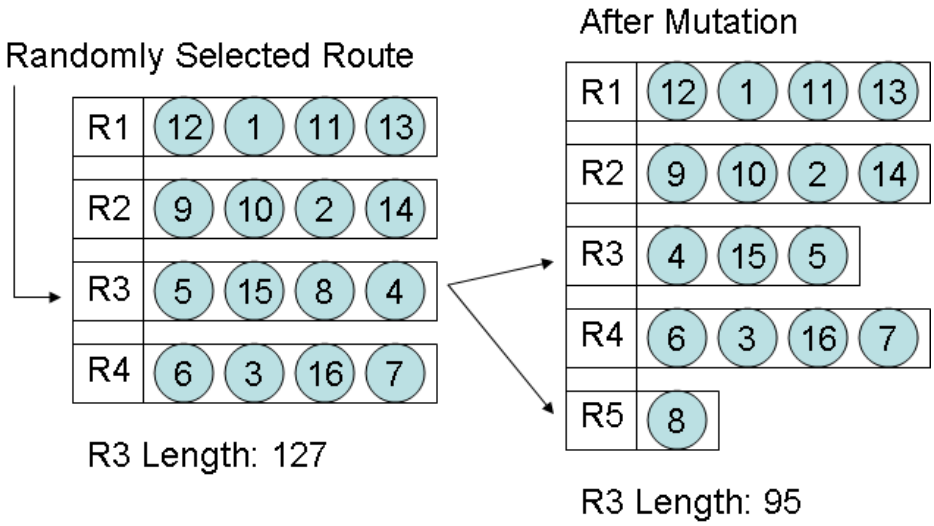

Figure 4.12: Best Route Cost mutation operator.

to the VRPTW chromosome which is only sensitive across a single route, making even slight changes can cause invalid solutions to be created.

4.9.1 Split Mutation. Split mutation randomly selects a route within the SRP solution and attempts to reduce the total length of that route be eliminating unnecessary target visitations. Each customer is satisfied with a certain number of UAVs at its location, however more can be present than are actually needed. This may cause a route to be longer than it needs to be since its divergence to an unnecessary target takes longer than a direct route. The split mutation operation determines if this is occurring in a random route and attempts to remove the target from the vehicles flight plan. If this operation then results in an infeasible solution it is considered to have failed, and is not implemented. This process is illustrated in Figure 4.13.

4.9.2 Vertical Swap Mutation. The vertical swap operator swaps two different locations vertically in a given solution. This is in contrast to the VRPTW swap mutation in section 4.8.2 in which the swapped targets can be anywhere. Columns within the SRP have a close approximation to time within the solution. It is not exact because distance information is not contained within the solution, and cities in

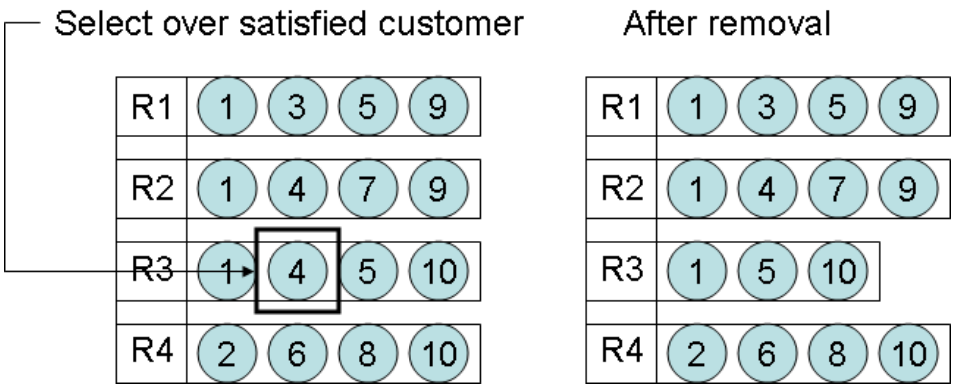

Figure 4.13: Split mutation operator for the SRP.

the same column may not actually be visited at the same time. The swap operator randomly selects a column and two different targets within that column. These targets are then swapped and feasibility is checked. An infeasible solution is not used. Figure 4.14 illustrates this process. Note the swap of customer four and six between vehicle two and four.

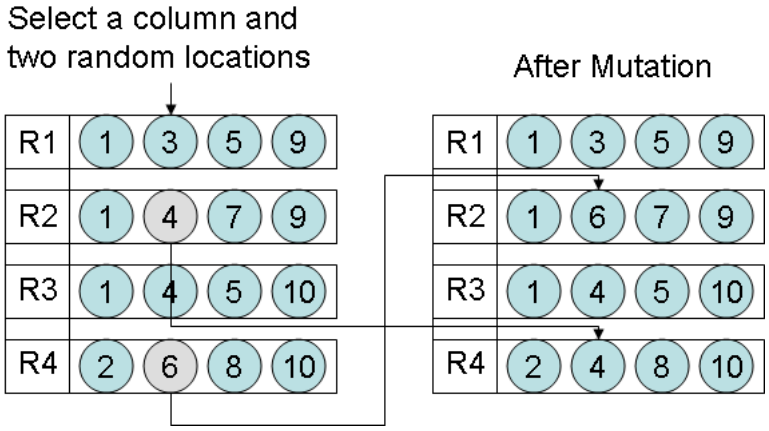

Figure 4.14: SRP vertical swap mutation operator.

4.9.3 Random Crossover with Tightening. The crossover operation must be done with particular care as effective alterations to the solution are difficult to achieve. The basic idea of the crossover operation is the same as the VRPTW crossover operation, from two solutions a random route is selected from each. This route is then added to the other solution. The problem is, unlike the VRPTW crossover operation,

subsections of a route can not be easily transferred between two solutions. In order to compensate for this, the route to be crossed is added to the solution as an entirely new route. The solution then undergoes an operation called tightening. During this operation the solution is searched to determine what customers are over satisfied or visited at inappropriate times, the customers in the new route are given preference for staying. The resultant solution contains the additional information of the crossover operation without the redundancy or errors the operation would otherwise result in. This process is illustrated in Figure 4.15.

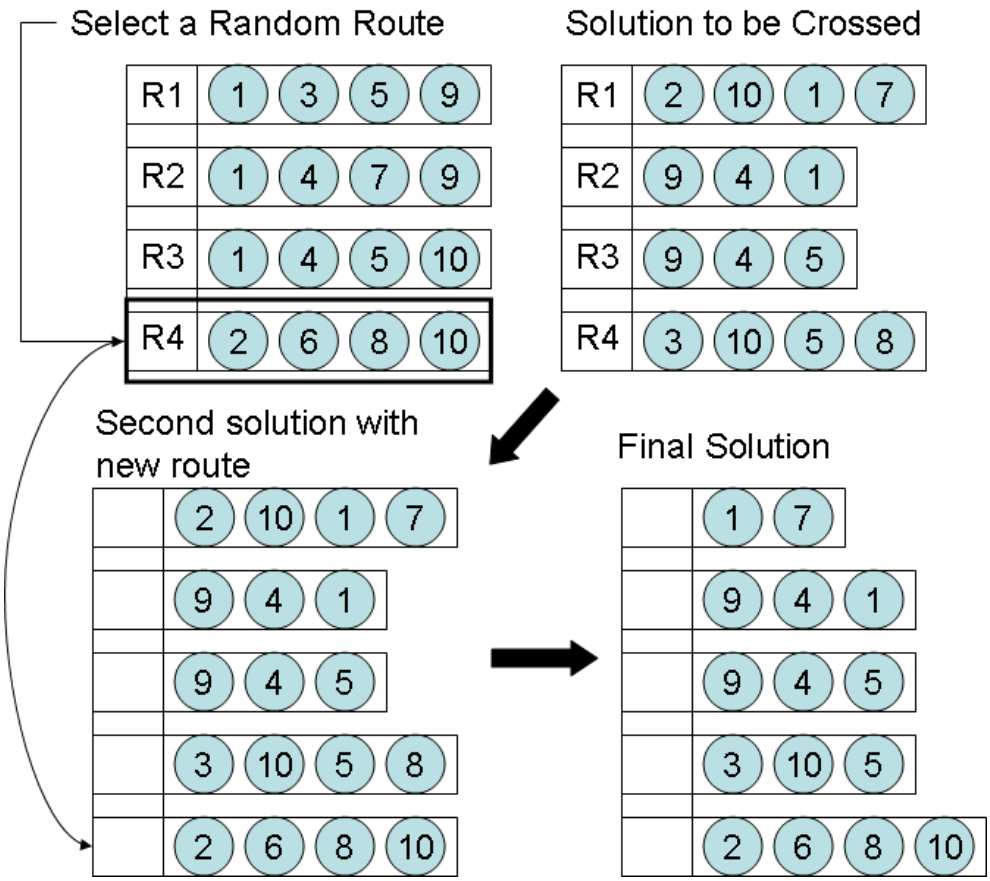

Figure 4.15: Crossover operator modified for use on the SRP.

4.10 Chapter Summary

This chapter contains the high level design for the MOEA solution to the VRPTW and SRP models defined in Chapter III. An evolutionary algorithm is chosen

due to success in its application from previous research and analysis of the problem of vehicle routing. Two selection methods and genetic operators for each problem are devised and detailed. In the next chapter it is shown how these operations are implemented into testable code.

V. Low Level Software Design

This chapter contains the low-level solution construction of a router for the VRPTW and SRP. The design presented in the previous chapter is mapped to an implementable software design. The programming language, data structures, and code infrastructure used are presented as part of the evolution of the research conducted.

5.1 Implementation Objectives

The high level GA design necessitates many elements of preplanning before construction of a solution can take place. These decisions include coding language decisions, the use of GA libraries, and the level of generalization required. The level of generalization implies how dependent the software is on the problem and how difficult it is to transition the software to different problems. The coded solutions for the VRPTW, SRP, and path planner consist of a series of processing steps: read in problem data, maintain customer(target) information, construct individual solutions, apply the MOEA to those solutions, and return the results in a readable format. The implementation must be able to accomplish these steps and return results compatible with the simulation software. The software must adhere to object-oriented standards, be generalized enough to allow for the alteration of algorithm parameters, and be compatible with available hardware.

5.2 Selection of an Evolutionary Computation Library

A basic structure exists within all genetic algorithm search techniques. This structure is defined by the transition of populations of solutions through a modification and selection process. Due to this structural concept it is advantageous to use a programming library or other software utility that has already been created with this basic structure in mind, hereafter refereed to as the infrastructure of the GA. There are a variety of infrastructure options available for evolutionary computation across many coding platforms. Some of these options are listed in Table 5.1.

Table 5.1: Evolutionary Algorithm Infrastructure Choices.

	Language	Object Oriented	Advantages	Disadvantages
GALib	C++	Yes	Previous work created in GALib. C++ data structure functionality.	Construction is very outdated and library is no longer supported by author. Functionality is limited.
Java GA Package	Java	Yes	Offers the portability of the Java engine. Java data structures and programming architecture easy work with.	Less efficient than a C based code. Not easy to integrate with previous work coded in C code.
Matlab GA Toolbox	NA	No	Matlab is easiest language to program in and toolbox functionality allows for fast construction.	Very slow processing and difficult to alter toolbox functionality past preset construction.
Open Beagle	C++	Yes	C++ data structure are very powerful and efficient. Very powerful functionality due to strict object oriented construction. Library is still currently supported by author.	Object oriented structure entails high level of complexity requiring longer initial production time.

For this research the Open Beagle (OB) library was selected. Previous routing work used the GALib library [41], however for this research it was determined that a transition to a more contemporary library would be beneficial. The OB library contains very powerful and well constructed tools for the creation of evolutionary algorithms. It is written in C++ allowing for easier integration with existing simulation uses, all of which are written in C++, and the library allows the use of the vector data structure. The library is written in very strict object oriented protocol, meaning little work is required on the part of the user to get program specific details inte-

grated into the overall programm structure, assuming they are written to the same Object Oriented (OO) standard. More details concerning the implementation level design aspects of OB can be found in Appendix B and online [18]. The selection of this infrastructure drives the code level requirement of all the program components as well as the data structures available (C++ data structures).

5.3 Software Design

The OB system is capable of a variety of different evolutionary computation methods including Genetic Programming (GP) and Evolutionary Strategies which arises from the object oriented structure of OB shown in Figure 5.1.

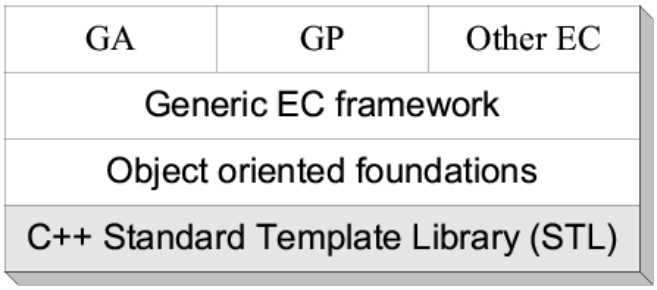

Figure 5.1: Open Beagle software architecture [17].

The top level GA structure contains the problem specific functions as seen in Figure 5.2. Note the use of different classes within the OB structure to separate the various operations. An expansion of each of these classes in the following sections shows their internal structure.

5.4 Replacement Strategy and Algorithm Structure

Within OB, the algorithm being used is implemented as a replacement strategy which selects individuals from the population of children and parents to be put into the next generation. As presented in Chapter IV, NSGA2 [14] and SPEA2 [62] are selected as the replacement strategies. Both algorithms are implemented in a similar fashion within the overall OB structure, with the only major difference being the ranking and selection functions used. The NSGA2 algorithm is installed as an option

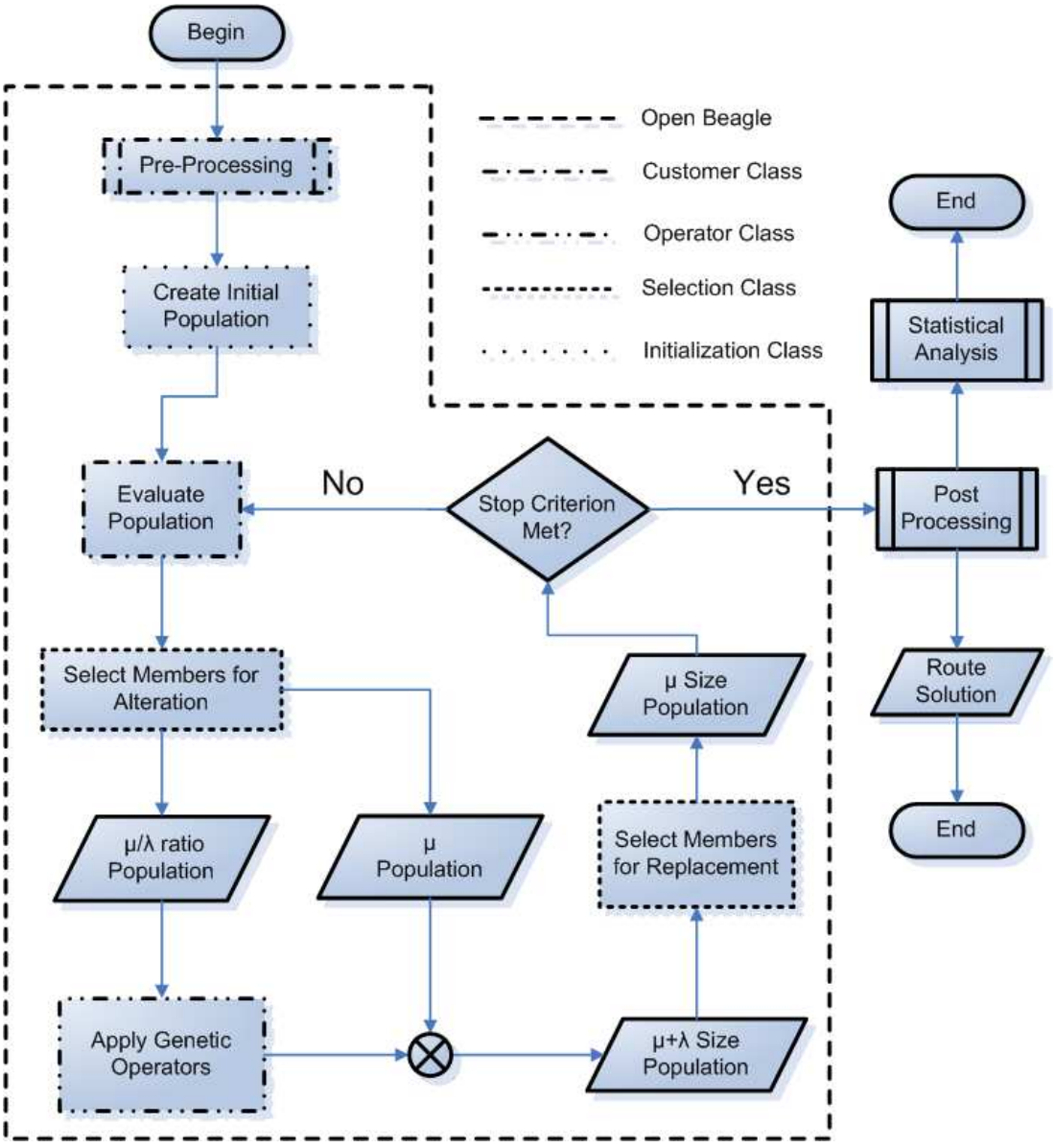

Figure 5.2: Evolutionary Algorithm Structural Implementation.

within the OB program and was written by Marc Parizeau. The SPEA2 algorithm is based on the coded algorithm written for the PISA system [5] and was translated into the OB form by the author.

Figure 5.3 describes both a *replacement* and *selection* method contained within the selection class. The overall EA structure in Figure 5.2 shows the purpose of both of these methods. The selection class contains both methods within its class.

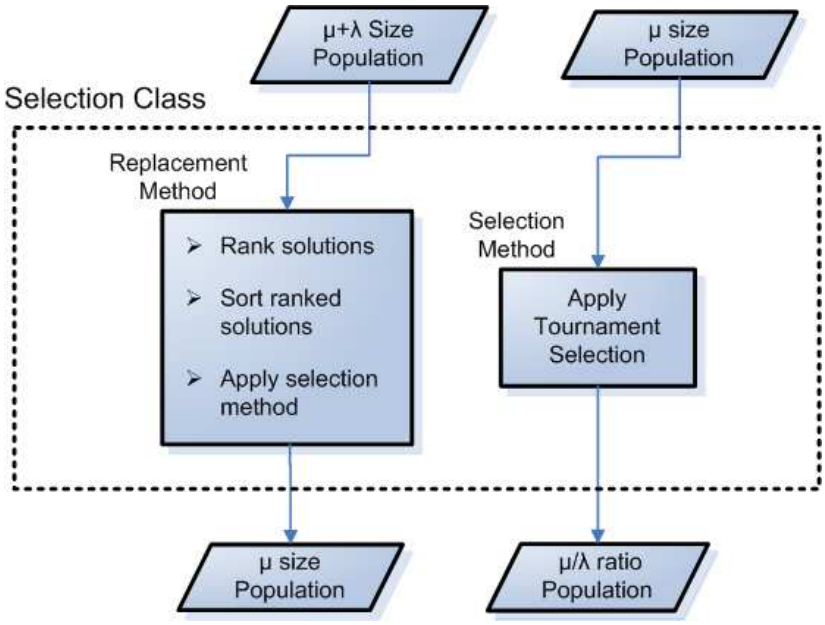

Figure 5.3: Selection class structure.

5.4.1 NSGA2. The NSGA2 uses two functions to rank and replace individuals: *sortFastND* and *evalCrowdingDistance*. The *sortFastND* function ranks all solutions by non-dominated rank. The sorted solutions are then put back into in the population via the *evalCrowdingDistance* method. Figure 5.4 shows the structural implementation of the NSGA2 algorithm.

The sorting function is called on line 9 and returns an ordered list of individuals. The structure that defines this list is a vector of integer vectors. The first vector in the set contains the rank zero, non-dominated members. The vectors that follow contain the remaining dominated individuals in ranking order (i.e. the second vector contains rank one). Following the sort and insertion of non-dominated members into the population (lines 8 - 15), the last pareto front is inserted into the population using the crowding distance function. Note how changes to the population are made by overwriting existing members (line 25 - 27).

The algorithm begins by being passed the *ioDeme*, which a structure that contains populations of individuals. A copy is made of the population and all children are generated within the copy population (lines 4 - 5). The *sortFastND* function then

```
1  void NSGA2Op::applyAsReplacementStrategy(Deme& ioDeme, Context& ioContext)
2  {
3          // Generate a new generation of individuals, merged with the actual one.
4          Individual::Bag lOffsprings(ioDeme);
5      lOffsprings = generateNextGeneration();
6
7          // Fast non-dominated sorting, followed by insertion of the first Pareto fronts.
8          NSGA2Op::Fronts lParetoFronts;
9          sortFastND(lParetoFronts, ioDeme.size(), lOffsprings, ioContext);
10         unsigned int lIndexDeme=0;
11         for(unsigned int j=0; j<(lParetoFronts.size()-1); ++j) {
12                 for(unsigned int k=0; k<lParetoFronts[j].size(); ++k) {
13                         ioDeme[lIndexDeme++] = lOffsprings[lParetoFronts[j][k]];
14                 }
15         }
16
17         // Insertion of the last Pareto front, using crowding distance
18         Individual::Bag lLastFrontIndiv;
19         for(unsigned int l=0; l<lParetoFronts.back().size(); ++l) {
20                 lLastFrontIndiv.push_back(lOffsprings[lParetoFronts.back()[l]]);
21         }
22         NSGA2Op::Distances lDistances;
23
24         evalCrowdingDistance(lDistances, lLastFrontIndiv);
25         for(unsigned int m=0; lIndexDeme<ioDeme.size(); ++m) {
26                 ioDeme[lIndexDeme++] = lLastFrontIndiv[lDistances[m].second];
27         }
28 }
```

Figure 5.4: NSGA2 Code Structure

receives the new population and a vector for storing reference numbers. The original *ioDeme* is then modified by overwriting existing individuals with new ones from the *lOffsprings* structure based on ID numbers in the *lParetoFronts* structure, updating the population with the first rank of non-dominated solutions. The *evalCrowdingDistance* function is then used to evaluate the dominated members of the population. The remaining population members are updated with the results from this function.

5.4.2 SPEA2. The SPEA2 structure is the same as the NSGA2 structure, in that the individuals are generated and reinserted in the same manner. The difference is how they are selected for insertion. The code structure for SPEA2 is shown in Figure 5.5.

After a new generation is created the individuals are assigned a fitness value by the *calcFitness* function on line 11. Adjacency matrices are then generated at line 13. These matrices are used in truncation functions. The matrix data structure is implemented as another vector of vectors structure. Based on the number of non-

```
1  void SPEA2Op::applyAsReplacementStrategy(Deme& ioDeme, Context& ioContext)
2  {
3          // Generate a new generation of individuals, merged with the actual one.
4          Individual::Bag lOffsprings(ioDeme);
5      lOffsprings = generateNextGeneration();
6
7          //except for the first entry, IDs of zero indicate the indiv. was removed
8          SPEA2Op::vecFitness lIndivIDValues;
9
10         //calculate SPEA fitness values for all individuals
11         this->calcFitness(lIndivIDValues, lOffsprings, ioContext);
12         //create the distance matrices used in the truncation process
13         this->calcDistances(lOffsprings.size(), lOffsprings);
14
15         //truncate the population using the strength and fitness environmental selection method
16         if (fitness_bucket[0] > (*mArchiveSize).getWrappedValue())
17         {
18                 truncate_nondominated(outFitnessValues, inIndividualPool, ioContext);
19         }
20         else if (inIndividualPool.size() > (*mArchiveSize).getWrappedValue())
21         {
22                 truncate_dominated(outFitnessValues, inIndividualPool, ioContext);
23         }
24         return;
25
26         //add the selected individuals back into the population
27         unsigned int row=0;
28         unsigned int lIndexDeme=0;
29
30         for(unsigned int k=0; k<lIndivIDValues[0].size(); ++k) {
31         //a zero value indicates it was removed from the population
32                 if(lIndivIDValues[0][k] != KILL_VALUE)
33                 {
34                         ioDeme[lIndexDeme] = lOffsprings[lIndivIDValues[0][k]];
35                         ++lIndexDeme;
36                 }//end if
37                 if(lIndexDeme>=ioDeme.size()) //stop if you fill up the deme
38                         break;
39         }//end for
40 }
```

Figure 5.5: NSGA2 Code Structure

dominated individuals and the archive size, a process of either truncating dominated or non-dominated members is pursued (line 16). Member truncation occurs via the process described in Section 4.5.2. The *lIndivIDValues* data structure initialized at line 8 is a vector which contains the ID numbers of all members in order. As individuals are truncated the ID number is then changed to a negative one. When individuals are inserted back into *ioDeme* the existence of this value indicates that the value should not be included.

5.5 VRPTW Solution Implementation

The VRPTW solution uses an object oriented structure divided into into 3 classes: the customer information class, genetic operator class, and initialization class as seen in Figure 5.2. The selection methods are defined in section 5.4.

The customer information class contains information about individual customers and the methods for evaluating individual solutions. The genetic operator class encompasses all the modifying operations that take place on an individual. The initialization class contains the method that creates the first population. This section concludes with a description of the heuristic mutation procedure; Best Cost Route Mutation. The remaining operations are described in Chapter IV.

5.5.1 GVR Data Structure. As presented in Chapter IV, Genetic Vehicle Representation (GVR) defines the structure of a single solution. The requirement imposed by GVR is a data structure that contains each route, and within each route, each customer, in order. These requirements are met by utilizing the vector data structure available within the C++ language, specifically, a vector of integer vectors, shown in Figure 5.6.

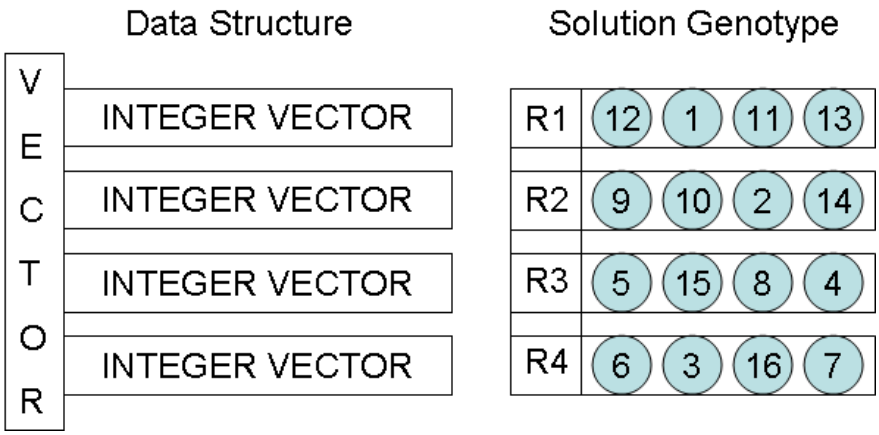

Figure 5.6: GVR Data Structure and Genotype for VRPTW.

The first vector contains a set of vectors, each of which represents a single route. Each of those vectors contain integers which represent identification numbers

for customers. The utility of this method is the use of iterators and standard vector operations that exist within C++. No customer information is kept within an individual, the individual is only a code of integers that correspond to information kept within the *customerInfo* class described in the next section.

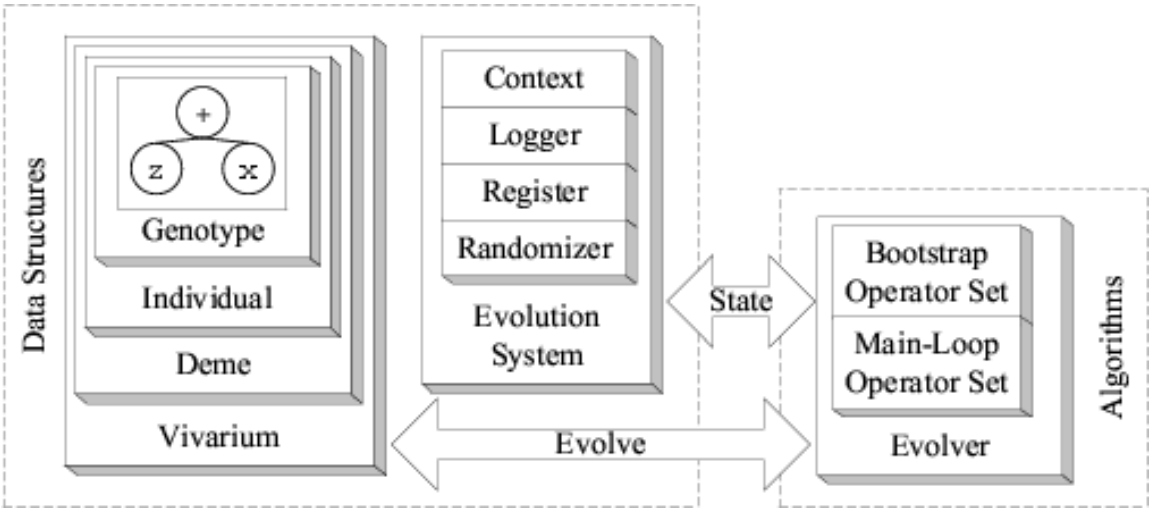

Figure 5.7: Individuals placement within the OB system.

Open Beagle stores each individual as a sub class member in a population hierarchy as shown in Figure 5.7. The utility of this structure allows populations and individuals to be passed in a completely encased class with the individual genotypes referenced as sub-class members. This not only keeps the design in line with software design principles but simplifies the construction of the GA by generalizing how the solutions are passed between different operation.

5.5.2 Customer Class. The customer class contains solution evaluation methods and customer relevant data. The evaluation methods are called by all levels of the algorithm as every solution must be checked for validity during construction and alteration which is why the methods are contained in a separate class, making it accessible to all other classes. The data structures of the customer class contain problem specific data used in the evaluation process and for heuristic calculations in some mutation operator methods (i.e. Best Cost Mutation). Figure 5.8 shows this structural concept.

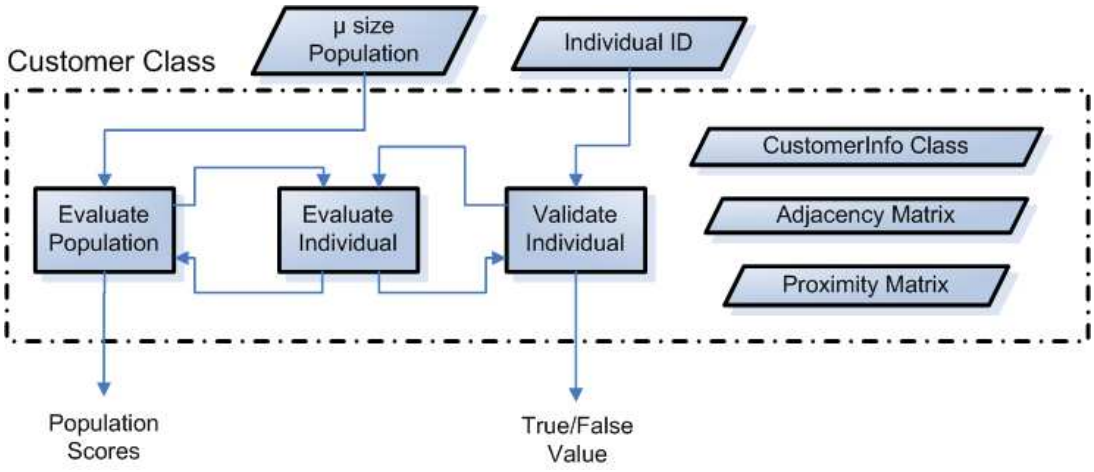

Figure 5.8: Customer class structure.

As problem data is read in, customer objects are created. Each object has problem information associated with it (ID number, x coordinate, y coordinate, demand, ready time, due time, and service time). These objects are stored within a vector that is kept by the *customerInfo* class. A single *customerInfo* object is created at startup which is persistent throughout the run of the GA.

Two important data structures are used within the Customer class. These are the adjacency matrix [10] and the proximity matrix. Both are created by defining a vector of integer vectors. The adjacency matrix contains distances between all customer locations (i.e. edge costs). The purpose of the matrix is to serve as a lookup table for distance values during the evaluation procedures, alleviating the need to constantly recalculate distances.

The proximity matrix contains an ordered list of the customers by proximity. The row the vector is in implies the customer it is associated with. The proximity matrix is used to make the process of rearranging customers and determining better routes a more efficient process. Instead of repeatedly searching through the adjacency matrix to determine the closest customer a single search through a row of the proximity matrix returns the same information.

The customer class also contains methods for the evaluation of solutions and the verification of individual routes. These methods are not only used to determine the path length of an individual, but also to validate that changes made to a route have not resulted in an invalid solution. Two methods evaluate entire solutions; *evaluateSolution*() and *validateSolution*(). The evaluation method returns the actual values associated with the solution while the verification is a boolean method that returns true if the solution is valid and false if not. Each of these methods employ route level methods called *evaluateRoute* and *validateRoute*. The *evaluateRoute* method determines the total length and wait time of a route. *validateRoute* is a boolean method which calls the *evaluateRoute* method and ensures the values are within problem boundaries. The solution evaluation methods are passed entire solutions (vector of integer vectors) while the route level methods are passed only integer vectors. This break up of the methods is done in accordance with software coding principles.

The design impact of the customer class is to keep all problem specific data outside of the actual genetic algorithm. This reduces the amount of information that needs to be kept track of by the infrastructure. It also ensures a generalization of internal GA functions by requiring calls to the *customerInfo* class rather than putting the functionality in a single isolated GA function. New problem types or evaluation procedure modifications can then be more readily implemented.

5.5.3 Population Initialization. Before operations can be performed on a population, that population needs to be created. The initialization process creates a set of random population members, however the process is conducted in a more intelligent way than simply assigning random customers to vehicles. This process as well as its placement within the initialization class is shown in Figure 5.9.

To begin, a vector is assigned a random series of numbers the size of the number of customers. Each number represents a customer ID. The first individual in the list is added to the first route. To select the customer that follows, a check is made to the proximity matrix. The function goes through this list attempting to find the closest

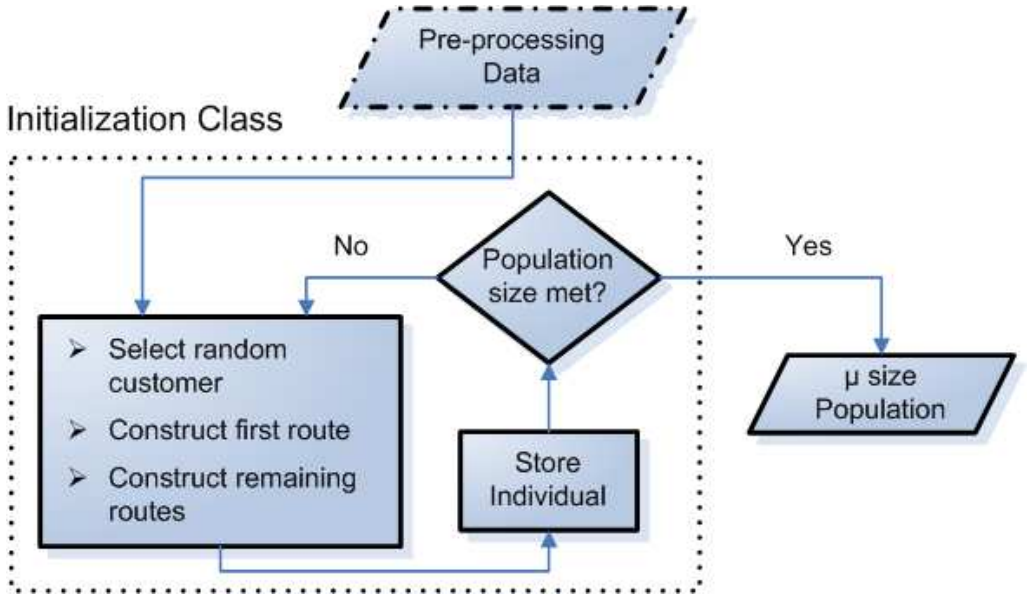

Figure 5.9: Population Initialization class structure.

customer that can be feasibly added. When no new customers can be added via the proximity matrix selection process, a new customer is taken from the random vector to add to the next route. It is of course ensured that through this process the random vector is updated as to which members are removed. This process is repeated until all solutions for the population are created.

5.5.4 Best Route Cost Mutation. The best route cost mutation begins by first selecting a random route within the solution to be optimized. Four vectors are then created, a route vector, a live copy vector, a proximity listing vector, and a transfer list vector. The route vector stores the route as it is being constructed. The live copy vector stores the previous route, its members are deleted as they are added to the route vector. The proximity listing stores the proximity listing for the customer currently being searched for. The transfer list vector stores customers that can not be added to the solution, these customers are later added into new routes.

A double nested for loop determines which of the customers in the route is closest to the depot. This becomes the first city in the optimized route. An iterative process then commences where the search for the next customer in the route is determined by

searching through the proximity listing for that customer. A customer is taken from the proximity listing and added to the route vector. If the route is valid that customer is removed from the live copy and the proximity listing is updated to reference this new customer. If a customer within the live copy vector can not be added back into the route it is put into the transfer list. If the transfer list contains more than one customer the previously described process repeats, customers from the transfer vector are added into new routes based on proximity.

This process results in an optimization of the selected route and an optimal placement of the remaining customers. The operator class shown in Figure 5.10 contains only the mutation operator described here, however all operators adhere to this same I/O structure within the operator class.

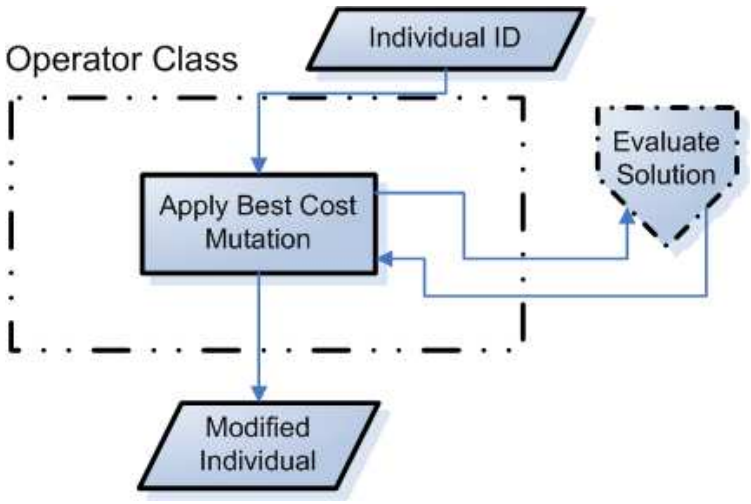

Figure 5.10: Operator class structure.

5.6 SRP Solution Implementation

The SRP implementation is a modification of the VRPTW implementation. All of the data structures within the VRPTW code are used in the same manner in the SRP code. This decreased the production time of the SRP solution. The SRP software structure also remains the same, containing a customer information class, genetic operator class, and initialization class. Each of the methods within these

classes are then modified for use the SRP solution. The overall structure is the same as that shown in Figure 5.2.

The customer information class contains information about individual customers and the methods for evaluating individual solutions. The genetic operator class encompasses all the modifying operations that take place on an individual. The initialization class contains the method that creates the first population. This section concludes with a description of the vertical swap mutation procedure implementation.

5.6.1 Modified GVR Data Structure. SRP uses the same chromosome structure as the VRPTW with the difference that customers do not appear only once. The implementation remains a vector of integer vectors. Each integer vector represents a single vehicles route within the solution and each integer in the route represents the ID of the customer to be visited.

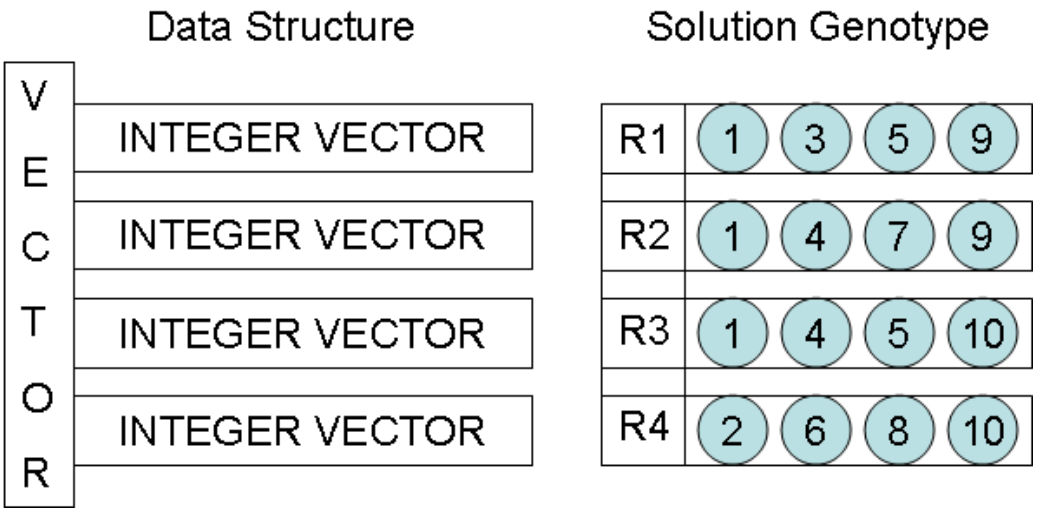

Figure 5.11: Modified GVR Data Structure and Genotype for SRP.

5.6.2 Modified Customer Class. The customer class contains the customer object and methods for solution evaluation. The information kept about a single customer is the same as for the VRPTW with the exception of customer demand. Since demand is no longer satisfied by capacity and instead by the number of UAVs on site, the demand values meaning is changed.

Solutions can no longer be evaluated by route since customer satisfaction occurs across all routes at once. Customer demand is also not not satisfied until all vehicles arrive, so wait time is not only based on earliest arrival but also latest arriving vehicle (wait time equals earliest arrival time minus latest arriving member time if latest member arrives past earliest start time). Due to this, evaluation take place by column across all routes at once.

The process of evaluation works by first determining the path length from the depot to the fist customer via the adjacency matrix. The cost value is kept in a vector the size of the number of vehicles in the solution. The method then iterates through the solution updating the cost vector for each additional customer. Wait times are calculated by determining the vehicle arriving last. The method ends by connecting all routes back to the depot to determine the final path length and cost for each vehicle.

5.6.3 Population Initialization. The initial population for the SRP is created using the same heuristic method as described for the VRPTW. A random customer is added to a route followed by the next, closest feasible customer. The proximity is determined via the proximity matrix. The modification for the SRP stems from the need to track all routes as they are being constructed. The process is shown in Figure 5.12.

The fist selected customer is added to the number of vehicles needed to satisfy its demand. The process then iteratively attempts to add new customers to existing routes in the solution until the demand for that customer is satisfied. If the demand is not satisfied after all existing routes have been attempted a new vehicle is added to the solution. This process continues until all customers have been assigned.

5.6.4 Vertical Swap Mutation. To apply vertical swap mutation two random locations are selected within the individual, ensuring that they occur at the some point in time (i.e. the same column). To determine if a swap is possible a copy of the

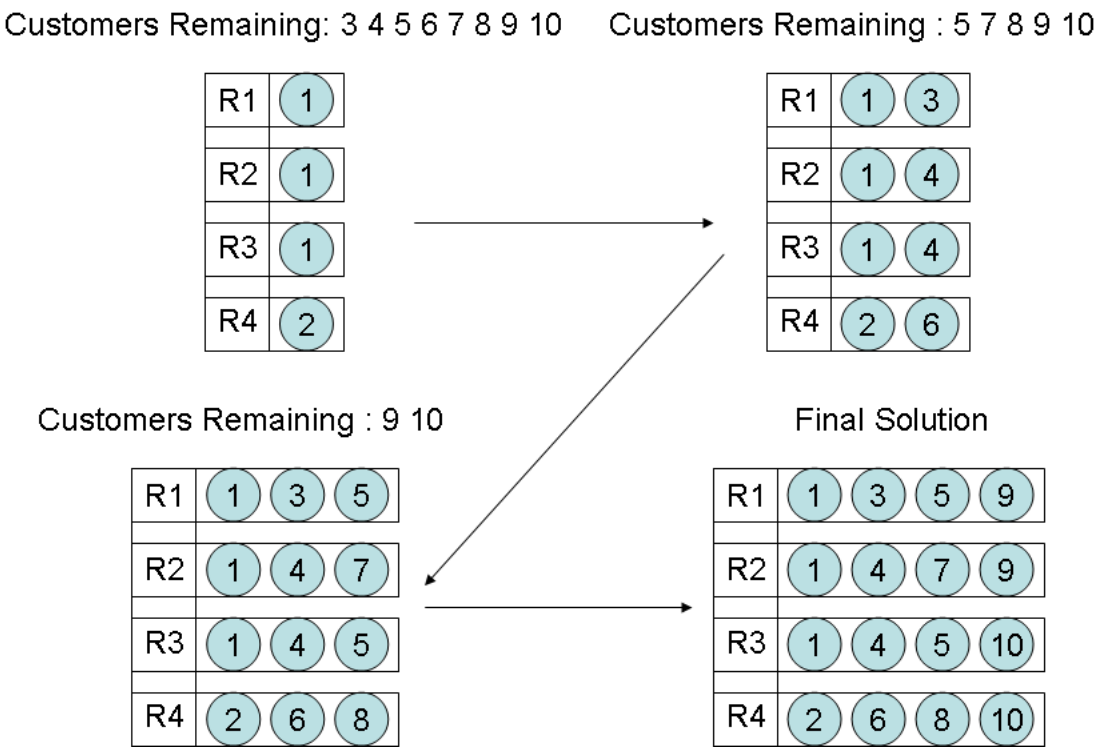

Figure 5.12: SRP Initialization Procedure Example.

solution is modified to include the swapped members. This solution is then passed to the customer class for evaluation. If the solution returns as valid the mutation is performed on the actual solution and returned to the population. If the swap has resulted in an invalid solution the mutation fails, if not, then the mutation is successful and the method terminates.

5.7 Chapter Summary

This chapter presented the software solution design. This design process entailed selection of EA infrastructure software, the separation of tasks within the program, and feasible implementation designs for the different genetic operators. Some of these aspects were driven by the infrastructure choice. The OB library has a very specific construction procedure for each genetic operator simplifying the implementation task. Outside the scope of the infrastructure is the customer class which separates all cus-

tomer data and evaluation methods from the main evolutionary algorithm. In the next chapter a series of experiments are designed to validate these design choices.

VI. Experimental Procedures

This chapter contains information about the design of experiments performed as part of this thesis. The objective of the experiment design phase is to develop testing methods for the problem model and solution design. The experiments divide into three sections which test the effectiveness of the routing software, the path planner, and the simulator. The purpose of these experiments is to validate the algorithm design of the routing software by applying standard VRPTW benchmark problems and comparing the solutions to best known solutions. Modified versions of these test problems are also applied to the SRP routing software in order for the results to be comparable.

6.1 Experimental Design Objectives

The goal of any experiment is to contribute evidence to the hypothesis proposed. In this case, there are two hypotheses to test stemming from the objectives defined in Chapter I; the proposed solution design for application to the VRPTW is valid across a spectrum of benchmark problems, and the solution design for application to the SRP produces valid results comparable to those obtained in the corresponding VRPTW benchmark. These objectives drive the experiment design such that a set of benchmarks are applied to the VRPTW and SRP solutions resulting in a set of valid solutions. These solutions then contain measurable metrics of total path length, total vehicle count, total wait time, and average path length (these metrics apply to both the VRPTW and SRP). In the case of the SRP the benchmarks are modified such that customer demand is an indication of vehicle count and not capacity demand, as in the VRPTW. Comparison of these metrics of performance allows for an intelligent comparison of the solution process to benchmark problems.

Accurate performance comparisons require the application of different design choices to the same problem, using the same settings when possible. To fulfill this requirement genetic algorithm settings are chosen and kept constant across the spectrum of algorithm choices. These settings are determined through empirical experiments

deemed to best represent the performance capability of the different genetic operators. Population size and generation limit are chosen within the desire to limit program run time.

Three different algorithm designs, each with two options for selection strategies, are used in the experimental procedures. These three designs are NSGA2, SPEA2, and a biased elitism algorithm. The biased elitism algorithm uses no strategy to rank solutions instead using an elitist ordering procedure that is biased toward path length. The top number of individuals, equal to the population size, are selected from the population after genetic alteration. Each of these designs is then paired with either a random or tournament selection process. Recall that selection refers to how solutions are selected for genetic mutation. Tournament selection means some number of random individuals is selected from the population, with replacement, and ranked (biased by path length) with the top rank selected for alteration. The SRP experiments employ only the use of the tournament selection method as random selection was deemed more harmful to the SRP solution process from the fact that the genetic operators employ no local search techniques.

6.2 VRPTW and SRP Experiments

The most commonly used benchmarks for the VRPTW are the Solomon problems developed in 1987 [45]. They exist in three different varieties; a random distribution of customers (R), clustered sets of customers (C), and hybrid (RC). Each of these three problems comes in dimensions of twenty five and fifty customers. In order to examine the effectiveness of the software as well as the impact of the multi-objective design, two problems from each type are tested, listed in Table 6.1. The use of this variety of problems illustrates the impact of problem type on the solution design as well as solution performance in different instances. The number designation of each problem constitutes the time windows that exist for that problem. Problems that begin with a one, such as R109, have small time windows, while R206 has much larger time windows.

Table 6.1: Solomon test problem selections (**Modified for SRP**).

	Random		Cluster		Hybrid	
25 Targets	R206	**R109**	**C103**	C205	**RC107**	RC202
50 Targets	R206	R109	C103	C205	RC107	RC202

Each test problem contains a set of target coordinates, target time windows, and vehicle capacity. The test file is structured as seen in Figure 6.1. The Euclidean distance between targets is considered to be the edge cost. The same problem selections are applied to the SRP solution modified in the demand column to ensure that each problem contains a realistic UAV requirement.

RC202

VEHICLE

NUMBER	CAPACITY
25	1000

CUSTOMER

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME
0	40	50	0	0	960	0
1	25	85	20	0	911	10
2	22	75	30	0	919	10
3	22	85	10	0	910	10
4	20	80	40	644	764	10
5	20	85	20	0	909	10
6	18	75	20	388	508	10
7	15	75	20	0	914	10
8	15	80	10	367	487	10
9	10	35	20	371	491	10
10	10	40	30	519	639	10
11	8	40	40	195	315	10
12	8	45	20	0	917	10
13	5	35	10	653	773	10
14	5	45	10	35	155	10
15	2	40	20	174	294	10
16	0	40	20	255	375	10
17	0	45	20	703	823	10
18	44	5	20	335	455	10
19	42	10	40	254	374	10
20	42	15	10	537	657	10
21	40	5	10	0	905	10
22	40	15	40	375	495	10
23	38	5	30	201	321	10
24	38	15	10	681	801	10
25	35	5	20	784	904	10

Figure 6.1: Solomon test file example of 25 dimension hybrid problem.

Algorithm effectiveness varies greatly as different parameters within the program are tuned. The settings for each algorithm type were determined from empirical analysis and literature review [33]. The operator percentage indicates the chance that operator is used on an individual during the alteration phase. The more effective operators are used more often while the random operators are used less. All the options for the algorithm used to solve the VRPTW problems are listed in Table 6.2, the option for the SRP algorithm are listed in Table 6.3.

Table 6.2: VRPTW GA Settings.

Operator	**Setting**
Random Crossover	40%
Swap Mutation	25%
Inversion Mutation	25%
Insertion Mutation	10%
Best Route Cost Mutation	40%
SPEA2 Archive Size	80
Generation Limit	1000
Population Size	100
$\frac{\mu}{\lambda}$ ratio	2

The SRP software experiments use the NSGA2 and biased elitism algorithms. The reason for this is that results from the VRPTW reveal a consistent dominance of these two methods over SPEA2. Each algorithm/problem experiment is run thirty times in order to ensure reliable statistical analysis. Each replacement strategy uses a tournament selection method. The population size and operator application percentages are different from the VRPTW settings in order to counter the SRPs fragile structure. More simple operations are performed to take the place of a few intelligent operations. Experiments are run against a small subset of the problems applied to the VRPTW (those entries bolded in Table 6.3).

Table 6.3: SRP GA Settings.

Operator	**Setting**
Random Crossover	50%
Split Mutation	25%
Vertical Swap Mutation	5%
Generation Limit	5000
Population Size	100
$\frac{\mu}{\lambda}$ ratio	2

6.3 Testing Environment

Experiments are performed on an Opteron 248 processor operating at 2.2GHz. The system has 4 GB off chip memory and a 128 KB L1 cache. The code is compiled in 64-bit using GNU C++. All test programs are run using a bash scripting method and Portable Batch System (PBS) submission system.

6.4 Chapter Summary

This Chapter justifies the experiments performed using the VRPTW and SRP design developed in chapter V. A selection of VRPTW problems representing the spectrum of available benchmarks are used on both the VRPTW and SRP, with the problems for the SRP modified slightly in terms of of what the demand per customer represents. In the next chapter the results from these experiments are analyzed.

VII. Results and Analysis

This chapter contains results obtained from experimental procedures in Chapter VI, analytic analysis of the results and their contextual meaning. The VRPTW section contains results from experimental trials compared to literature results. A statistical comparison of these results further compares the performance of the different procedures used. Results from the SRP exist solely in the context of this investigation and are shown only in terms of the results obtained. Statistical analysis further compares the procedures used.

7.1 VRPTW Results

VRPTW optimization occurs across three dimensions of total path length, total wait time, and number of vehicles used. Previous analysis, and the classical view, of this problem attempts to optimize path length and the number of vehicles used [33] or path length alone [51]. Experiments performed in this investigation do not yield a single solution optimized in any one direction but rather a Pareto front of non-dominated values. In order to compare the results found here to those in the literature they are first shown in terms of the best path length found overall. The following box plots show the best path lengths available at the time of this writing compared to a distribution of values found from experimental trials (30 trials). Each box plot shows results for a single problem across six algorithm settings: SPEA2, NSGA2, Biased Single Objective; each of which uses either random or tournament selection. The wording used to express each of these settings is shown in Table 7.1. Each plot also shows the best answer for path length optimization found in Toth [51] and Diaz [12].

Where appropriate, a Kruskal-Wallis [1] test is used to further analyze performance for the different algorithms on specific problems. Following these box plots, solution space plots are shown across dimensions of path length and wait time in order

[1] A Kruskal-Wallis test is a variance analysis tool used to test the equality of a population among median groups. The result of the test is a window over the values of the distribution indicating if other samples are significantly different. Here, Matlab is used to create a visual of this window showing which sample are different from each other using an alpha of 0.05.

Table 7.1: Box plot label explanations for VRPTW experiments.

Plot Definition	**Meaning**
SPEA2 Tourn	SPEA2 replacement strategy using tournament selection for genetic operator application
NSGA2 Tourn	NSGA2 replacement strategy using tournament selection for genetic operator application
Bias Single Tourn	Biased Single Objective replacement strategy using tournament selection for genetic operator application
SPEA2 Rand	SPEA2 replacement strategy using random selection for genetic operator application
NSGA2 Rand	NSGA2 replacement strategy using random selection for genetic operator application
Bias Single Rand	Biased Single Objective replacement strategy using random selection for genetic operator application

to better examine algorithmic performance. The drive for this is that observing only path length can be misleading when examining MOEA performance. This section also divides into each type of problem defined in Chapter VI: random, cluster, and hybrid.

7.1.1 Random Distribution Problem. The difference in performance between high and low dimension problems is readily observable by comparing Figures 7.1 and 7.2. NSGA2 is observed to return results closer to the best answer, followed by the biased single objective algorithm, with SPEA2 doing worst. A Kruskal-Wallis statistical analysis performed in Matlab confirms these visual observations as seen in Figure 7.3 and 7.4.

The performance observation per algorithm is repeated in the R206 problem. NSGA2 again manages to pull ahead in terms of the path length objective and along with the biased algorithm approaches the best solution in the higher dimension 50 customer problem in Figure 7.6. Observe the consistent convergence of solutions in Figure 7.5 for NSGA2 using tournament selection. The performance of NSGA2 is further clarified by the statistical plot in Figures 7.7 and 7.8.

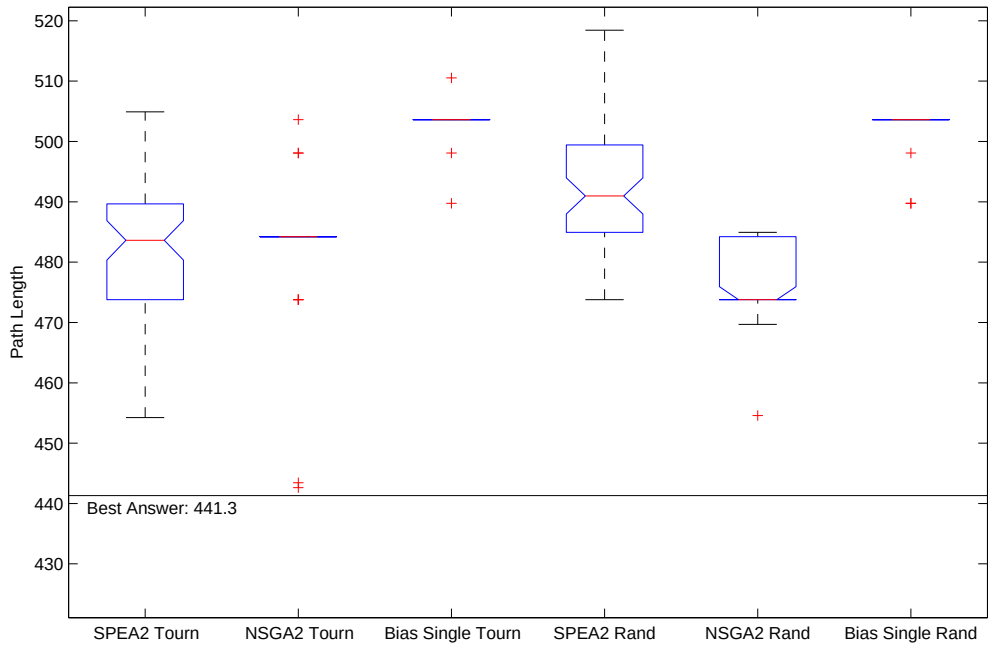

Figure 7.1: Trial results distribution for problem R109 with 25 customers

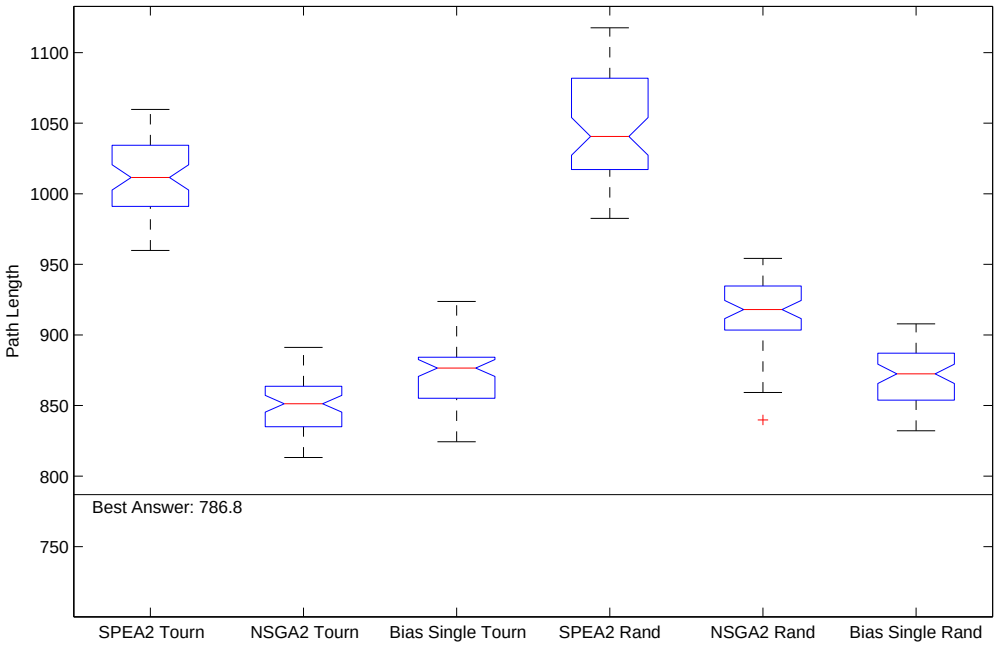

Figure 7.2: Trial results for random distribution problem R109 with 50 customers

7.1.2 Cluster Distribution Problem. Within the cluster benchmarks the path length objective becomes less consistent in returns. Figure 7.9 shows all methods closing in on the best answer with NSGA2 actually achieving it in a few trials.

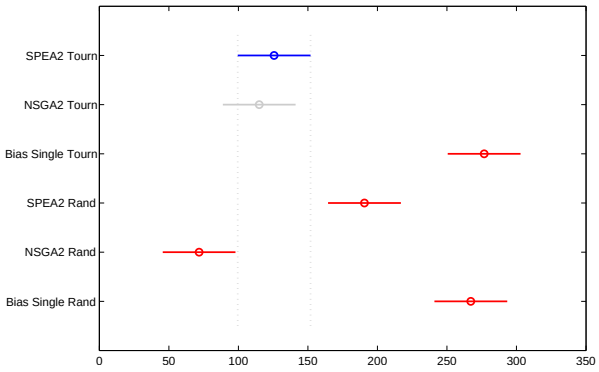

Figure 7.3: Significance plot for R109 with 25 customers

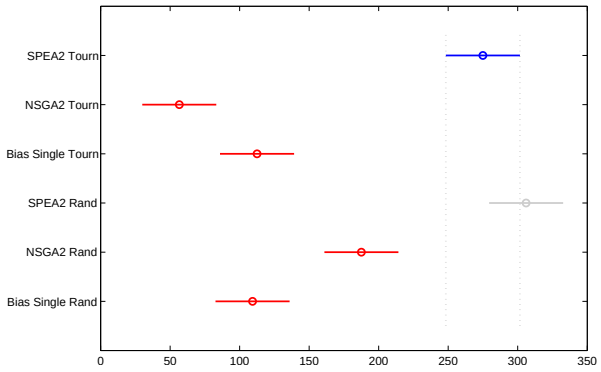

Figure 7.4: Significance plot for R109 with 50 customers

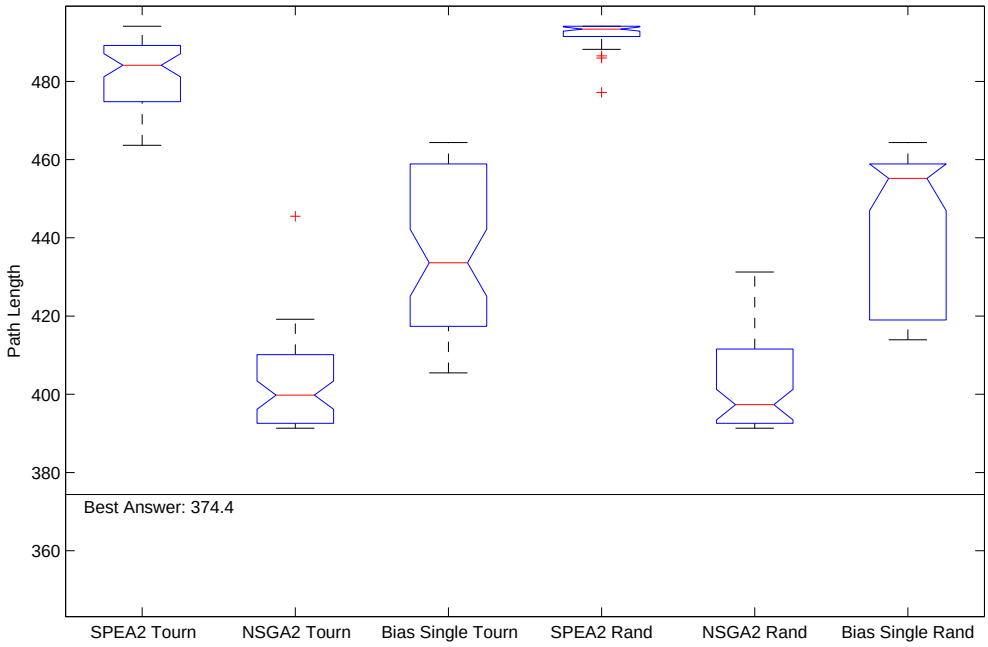

Figure 7.5: Trial results for random distribution problem R206 with 25 customers

Increasing the dimension of the problem, in Figure 7.10, causes a return to the performance seen so far, with no algorithm approaching the best solution. Statistical analysis shows only SPEA2 being significantly outperformed in Figure 7.11 and 7.12.

Figure 7.13 shows convergence using the biased algorithm with a wide dispersion of points using SPEA2 or NSGA2. Figure 7.14 maintains the decrease in performance over higher dimensional problems. Statistical analysis in Figures 7.15 and 7.16 show NSGA2 returning better results. It is interesting to note the results of NSGA2 with

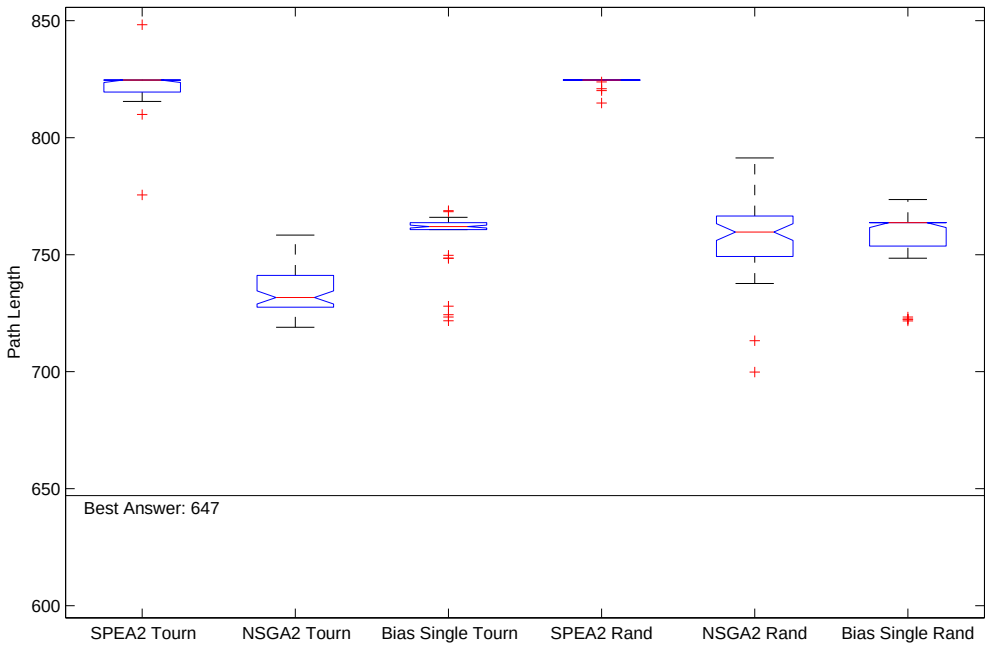

Figure 7.6: Trial results for random distribution problem R206 with 50 customers

random selection returning with such consistent results. This can be most likely attributed to the nature of the cluster problems working well with the genetic operators used.

7.1.3 Hybrid Distribution Problem. The hybrid problem would seem to represent the most difficult landscape to work in, however Figure 7.17 shows that no algorithm had particular trouble arriving at the optimal solution. This is less

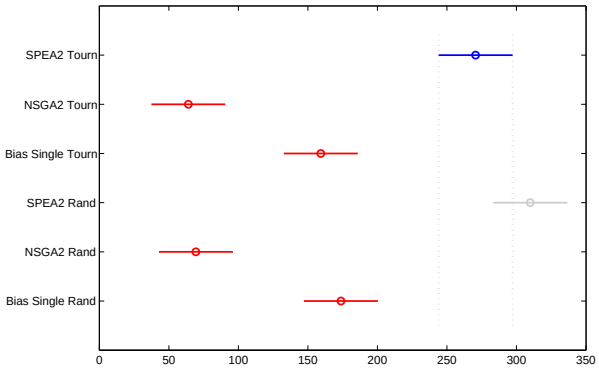

Figure 7.7: Significance plot for R206 with 25 customers

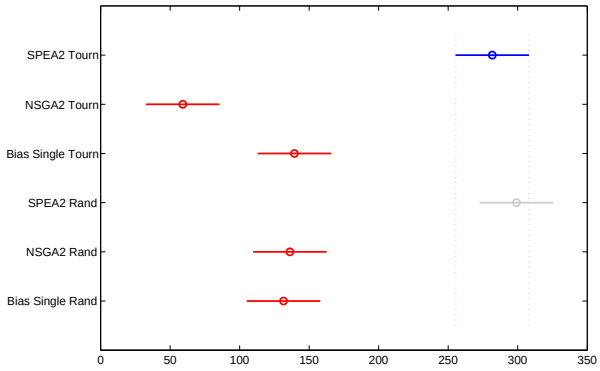

Figure 7.8: Significance plot for R206 with 50 customers

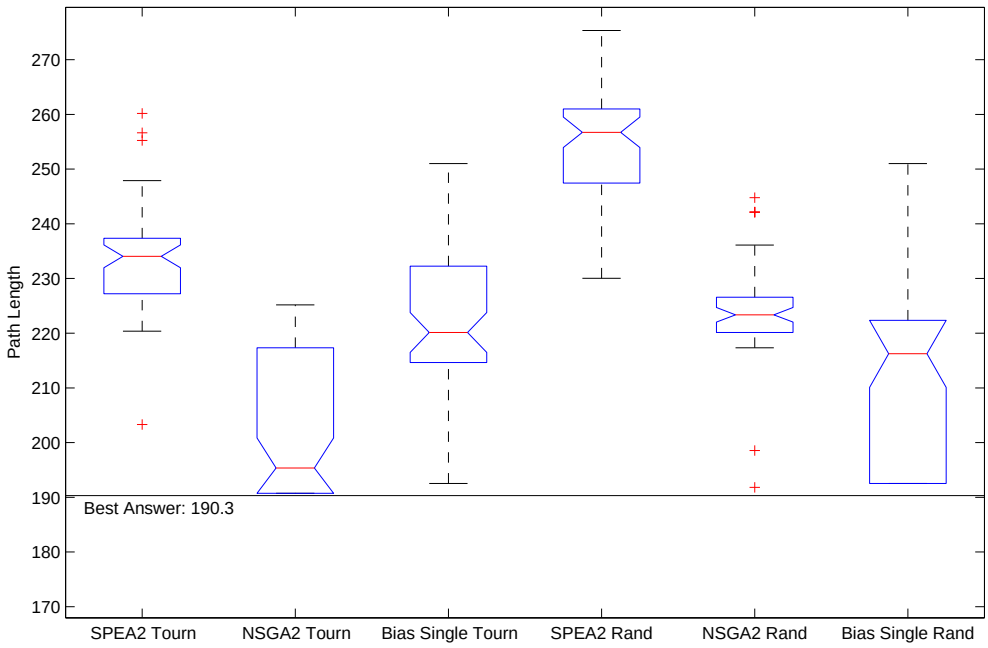

Figure 7.9: Trial results for random distribution problem C103 with 25 customers

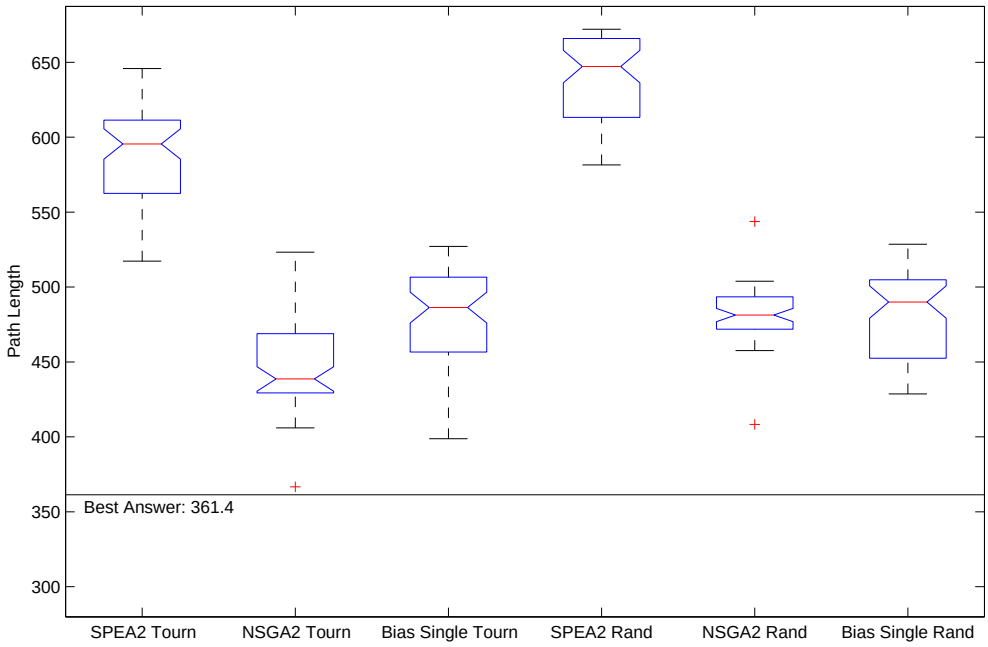

Figure 7.10: Trial results for random distribution problem C103 with 50 customers

true for the fifty dimension problem in Figure 7.18 as seen from the results being further from the optimal value line. Statistical analysis of the results in Figure 7.20 show NSGA2 and the biased algorithm returning consistent values with tournament

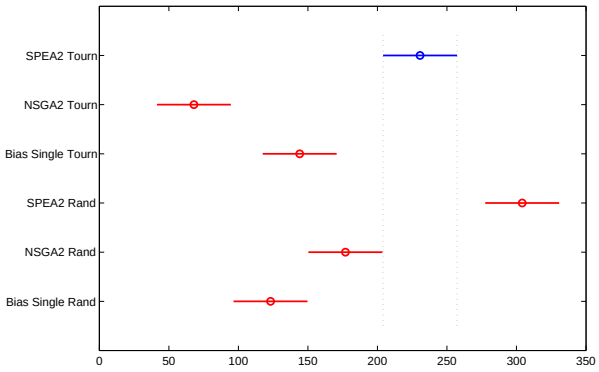

Figure 7.11: Significance plot for C103 with 25 customers

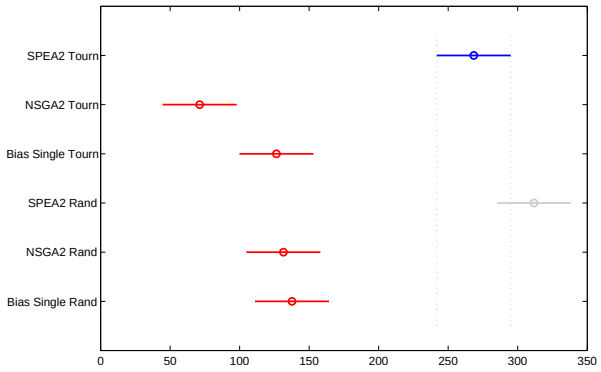

Figure 7.12: Significance plot for C103 with 50 customers

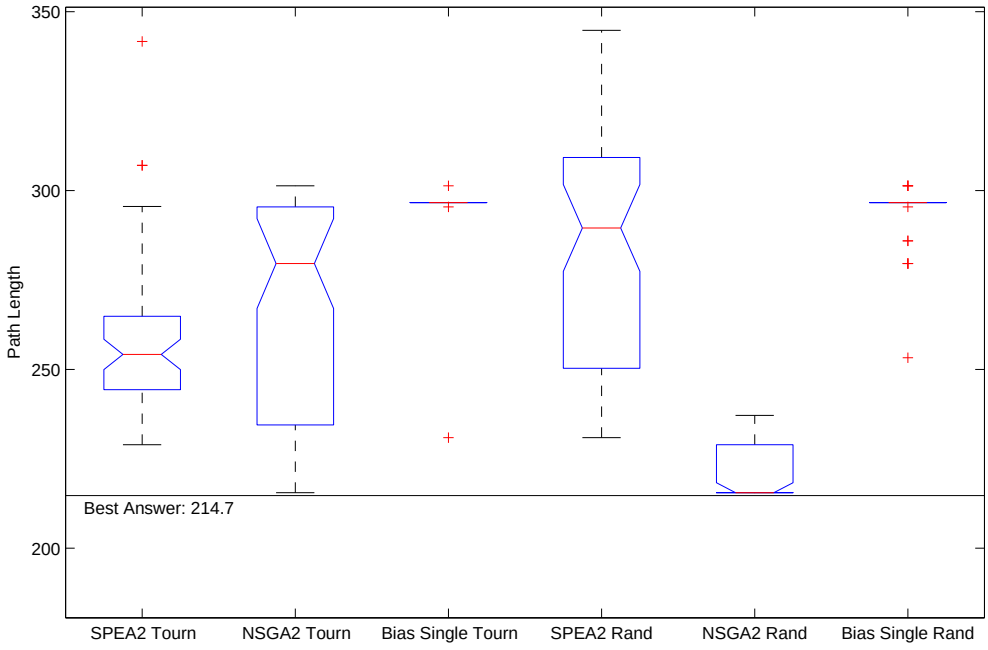

Figure 7.13: Trial results for random distribution problem C205 with 25 customers

selection being the deciding factor in superior performance. Further generational development would most likely force the solution closer to the optimal. Figure 7.19 shows NSGA2 achieving statistically better results by a small margin, further leading to the conclusion of its usefulness in developing solutions. However, previous results also show consistent returns using the biased algorithm, meaning no one strategy dominates overall.

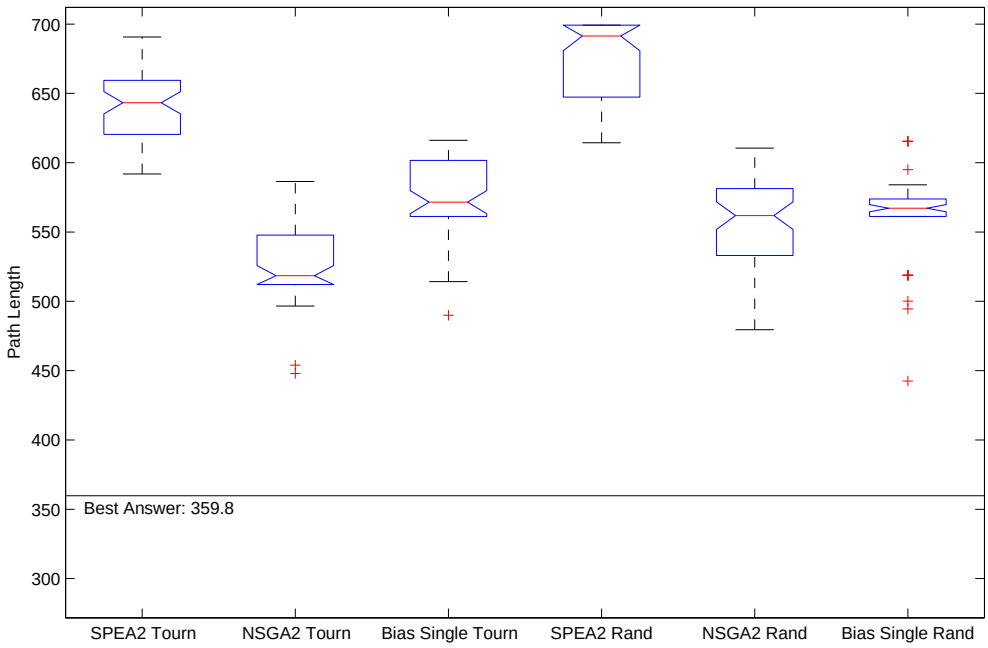

Figure 7.14: Trial results for random distribution problem C205 with 50 customers

The hybrid problem in Figure 7.21 again shows convergence of the biased algorithm while NSGA2 and SPEA2 maintain a larger coverage. This is also seen in Figure 7.22. The comparison plots in Figures 7.23 and 7.24 show consistent results across NSGA2 and the biased algorithm. SPEA2 is again beaten in this particular performance measure.

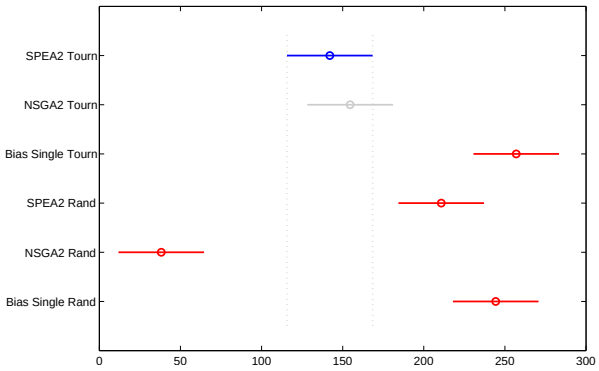

Figure 7.15: Significance plot for C205 with 25 customers

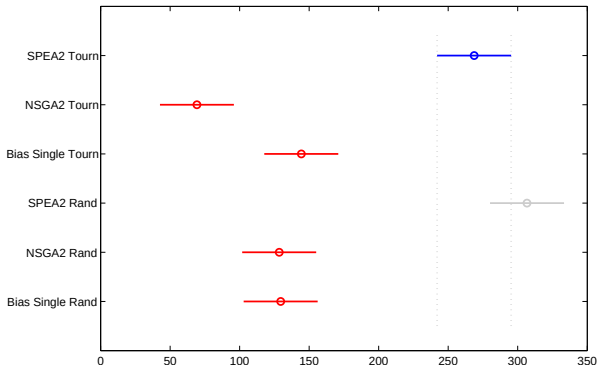

Figure 7.16: Significance plot for C205 with 50 customers

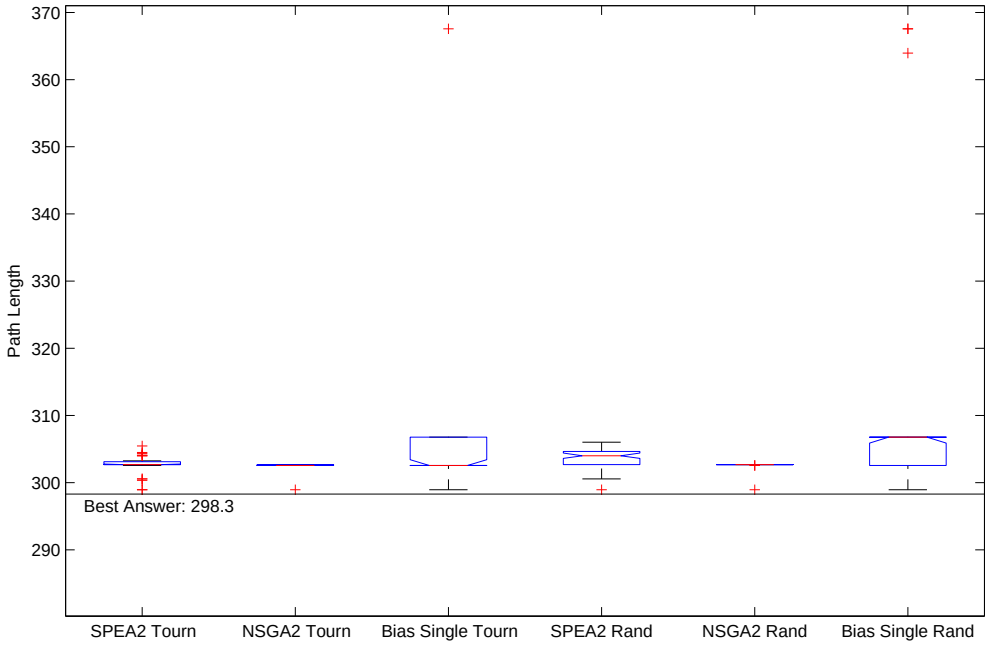

Figure 7.17: Trial results for random distribution problem RC107 with 25 customers

7.2 Impact of VRPTW results

Results from only observing the path length objective can be informative but also slightly misleading. It might be assumed that SPEA2 is being outperformed in all problem instances, and in terms of path length it is. However as seen in Figures 7.25-7.27 analysis of the non dominated front generated by the NSGA2 and SPEA2 trials shows a return of results consistent with what one would expect from a multi-objective problem.

The conclusion to be made is that the multi-objective solution is effective in returning a broad range of results and that these results are pushing the front of the problem. It is therefore not odd that the multi-objective approach did not return a near optimal value for path length. The returned value represents the solution space for the objectives selected. The fact that the returned values are close (i.e within 10 percent in most cases) to the highest benchmark value, shows the validity of the MOEA approach as being able to find the optimal value for a single objective, while

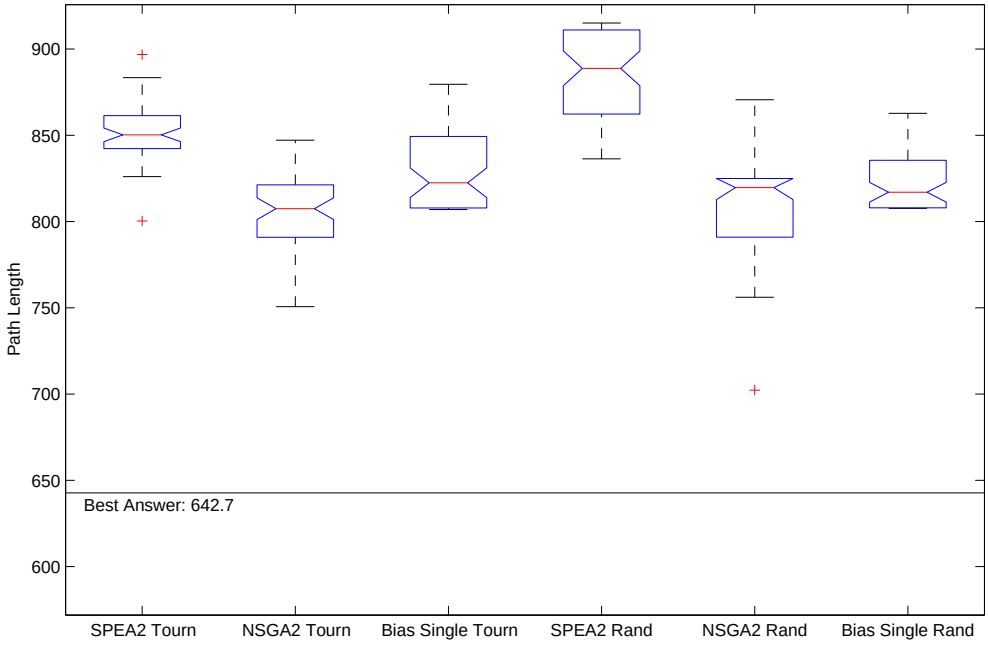

Figure 7.18: Trial results for random distribution problem RC107 with 50 customers

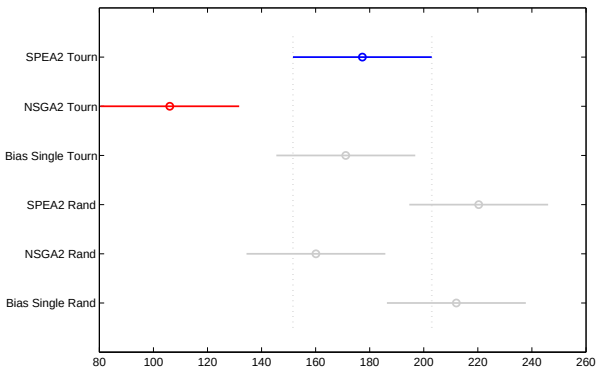

Figure 7.19: Significance plot for RC107 with 25 customers

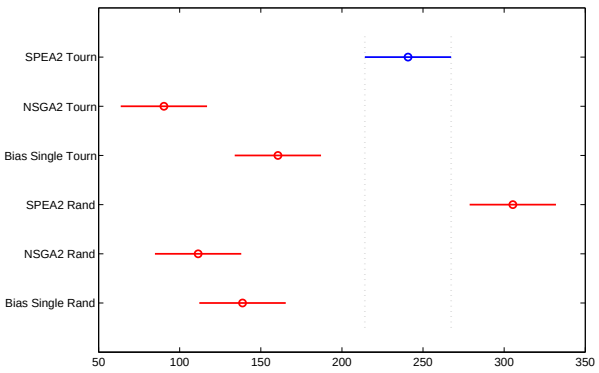

Figure 7.20: Significance plot for RC107 with 50 customers

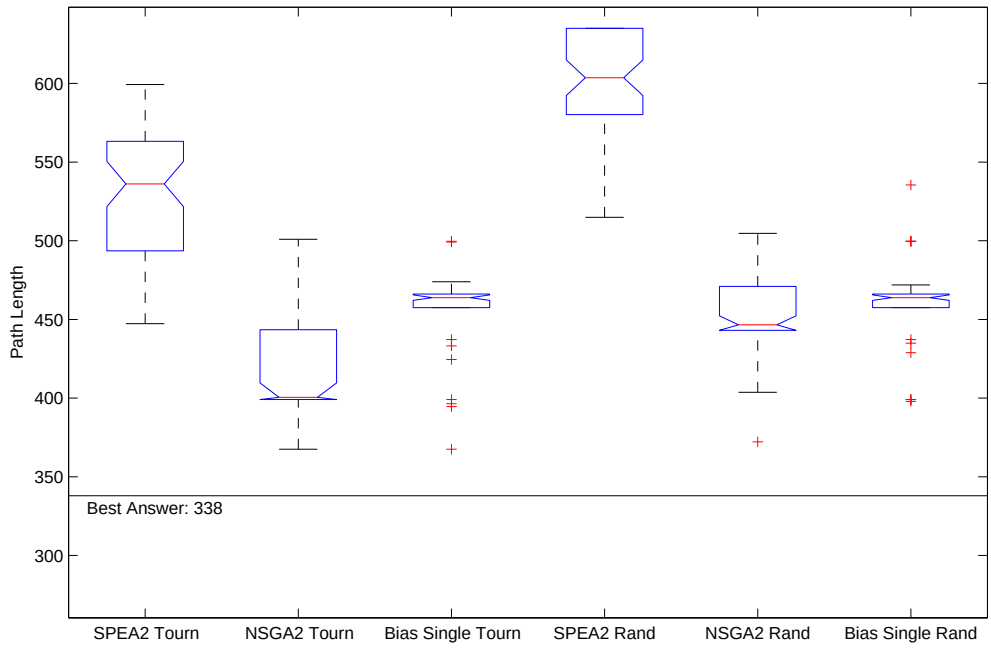

Figure 7.21: Trial results for random distribution problem RC202 with 25 customers

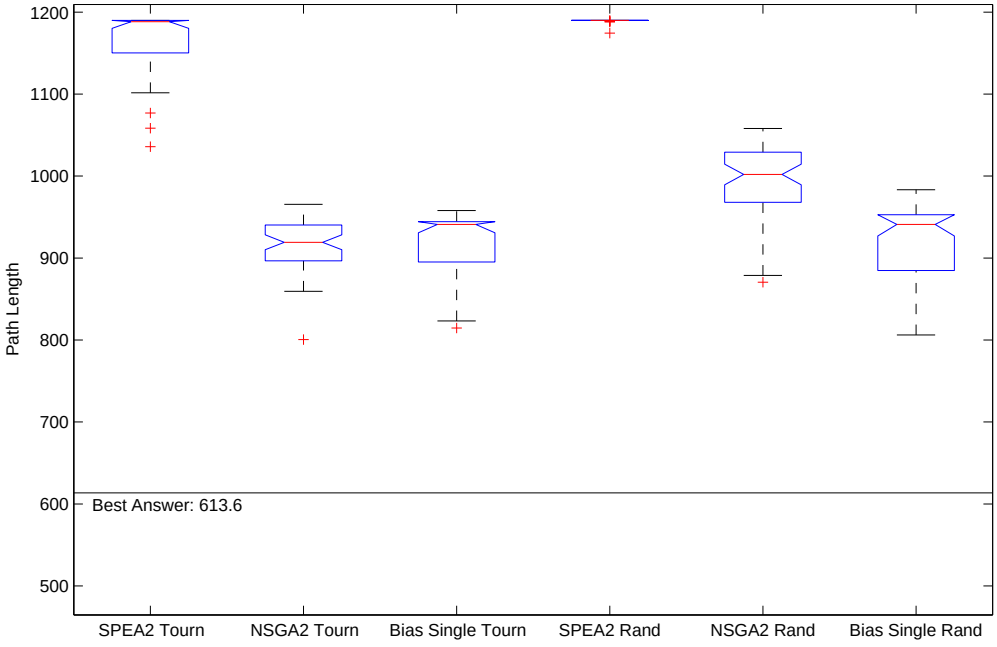

Figure 7.22: Trial results for random distribution problem RC202 with 50 customers

also optimizing across the range of objectives. In short, it appears that multi-objective optimization is appropriate for this particular routing problem.

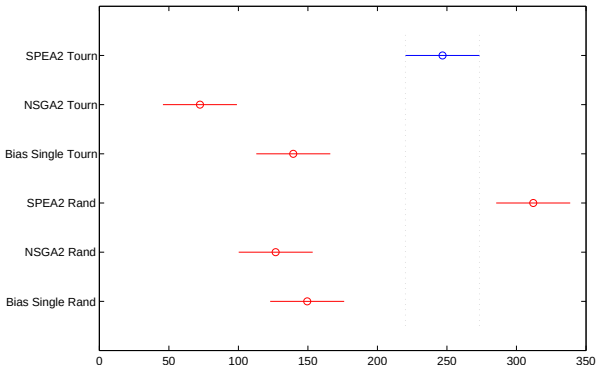

Figure 7.23: Significance plot for RC202 with 25 customers

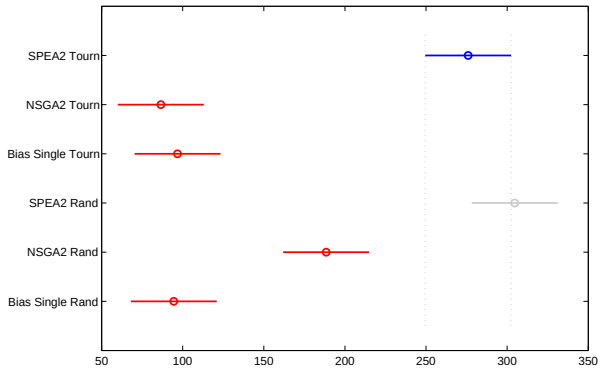

Figure 7.24: Significance plot for RC202 with 50 customers

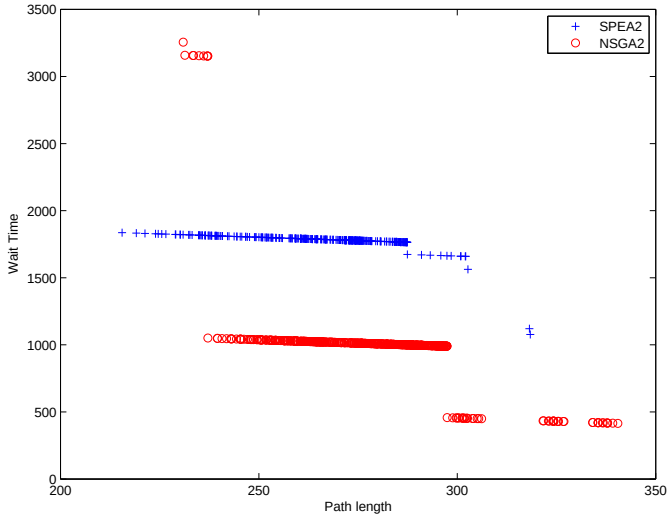

Figure 7.25: Non-dominated front comparing NSGA2 and SPEA2 for C205 with 25 customers

7.3 SRP Results

SRP optimization occurs across dimensions of total path length, wait time, the number of vehicles used, and average path length. As this problem formulation is unique to this investigation there are no readily comparable results. Though the problems differ in formulation the objectives remain the same between the SRP and VRPTW. As such, the results obtained from the VRPTW solution are compared in order to illuminate the hypothesis that the SRP represents a superior problem model in terms of individual vehicle operation and mission optimization. As in the previous section results are organized in box plots representing trial results for the

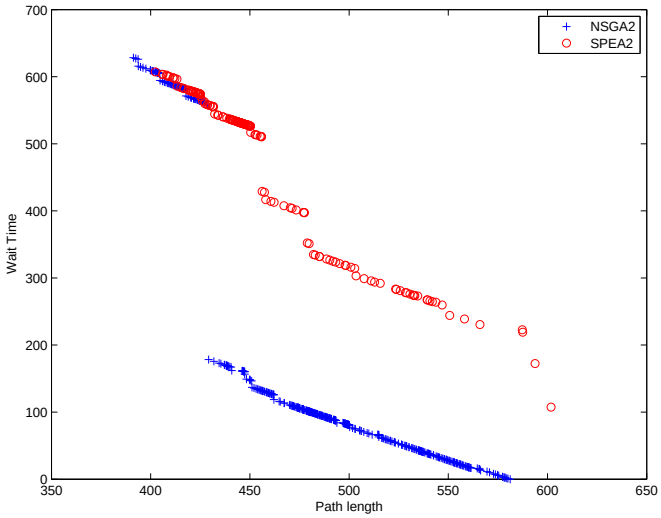

Figure 7.26: Non-dominated front comparing NSGA2 and SPEA2 for R206 with 25 customers

selected problem. Definitions for the labels used in the plots is found in Table 7.2 A comparison of the best results from the VRPTW solution are also shown both in terms of total path length and average path length (even though average path length is not an optimized objective in the VRPTW).

Table 7.2: Box plot label explanations for SRP experiments.

Plot Definition	**Meaning**
NSGA2 Tourn	NSGA2 replacement strategy using tournament selection for genetic operator application
Bias Single Tourn	Biased Single Objective replacement strategy using tournament selection for genetic operator application

7.3.1 Random Distribution Problem. Figure 7.28 compares the total and average path length returned by NSGA2 and biased algorithm. The biased algorithm seems to be converging while NSGA2 retains a larger spread of the solution space. In Figure 7.29 the average path lengths are compared to the average path length returns for the VRPTW using the same algorithm type. Neither NSGA2 or the biased algorithm perform significantly better than one another, as seen in Figure 7.30.

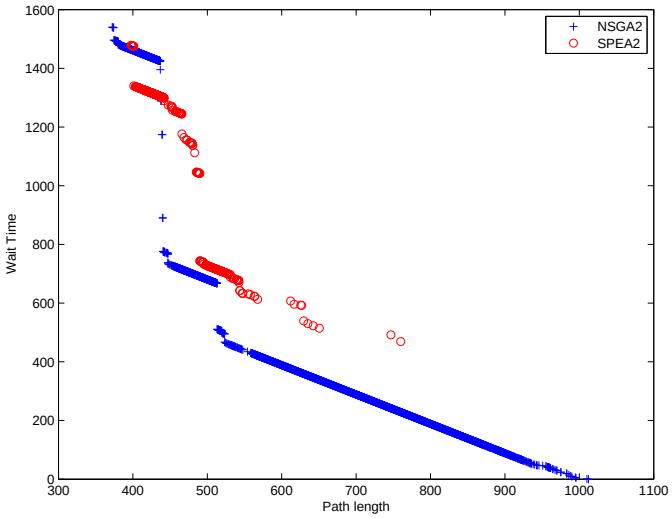

Figure 7.27: Non-dominated front comparing NSGA2 and SPEA2 for RC205 with 25 customers

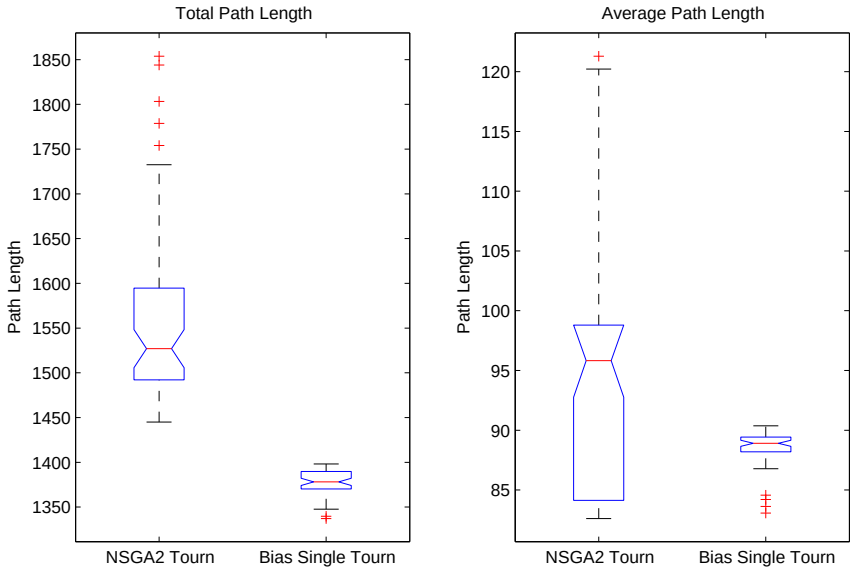

Figure 7.28: Trial results for random distribution problem RC107 with 25 customers comparing total path length and average path length per vehicle

7.3.2 Cluster Distribution Problem. For the cluster distribution problem, Figure 7.31 shows the return of total and average path length between NSGA2 and the biased algorithm. Further comparison between the same VRPTW results in Figure 7.32 shows a decreasing ability to handle this particular type of problem. This behavior should be expected as the VRPTW solution contains heuristic operators that

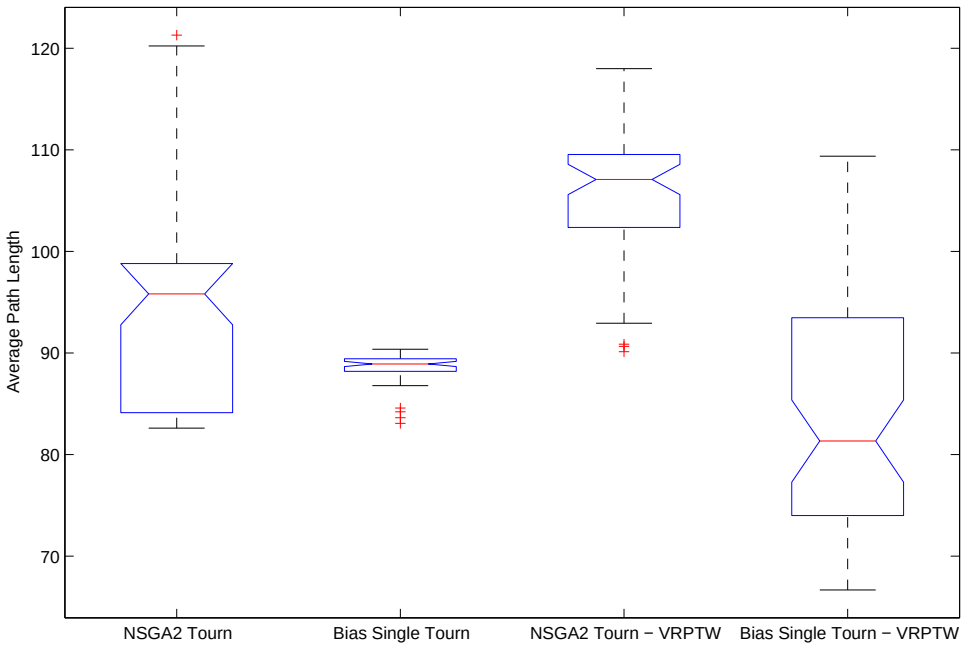

Figure 7.29: Comparison of average distance per vehicle in SRP results to VRPTW results for problem R109 with 25 customers

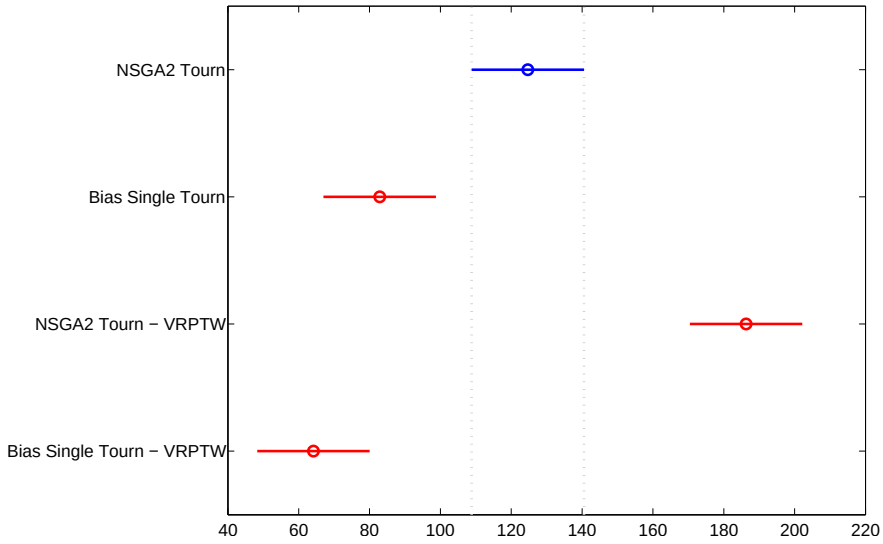

Figure 7.30: Statistical analysis comparing SRP results to VRPTW results for problem R109 with 25 customers

deal specifically with clustered targets, while the SRP does not. Note in Figure **??** the equal performance of the biased algorithm and NSGA2 for the VRPTW. Even with the high constraints of the SRP problem model it is still possible to return per vehicle path length of the same distance.

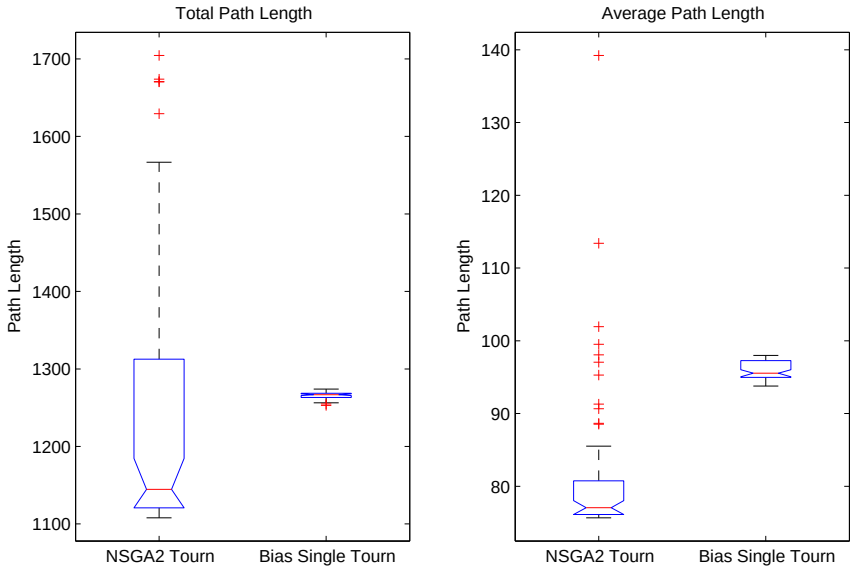

Figure 7.31: Trial results for random distribution problem C107 with 25 customers comparing total path length and average path length per vehicle

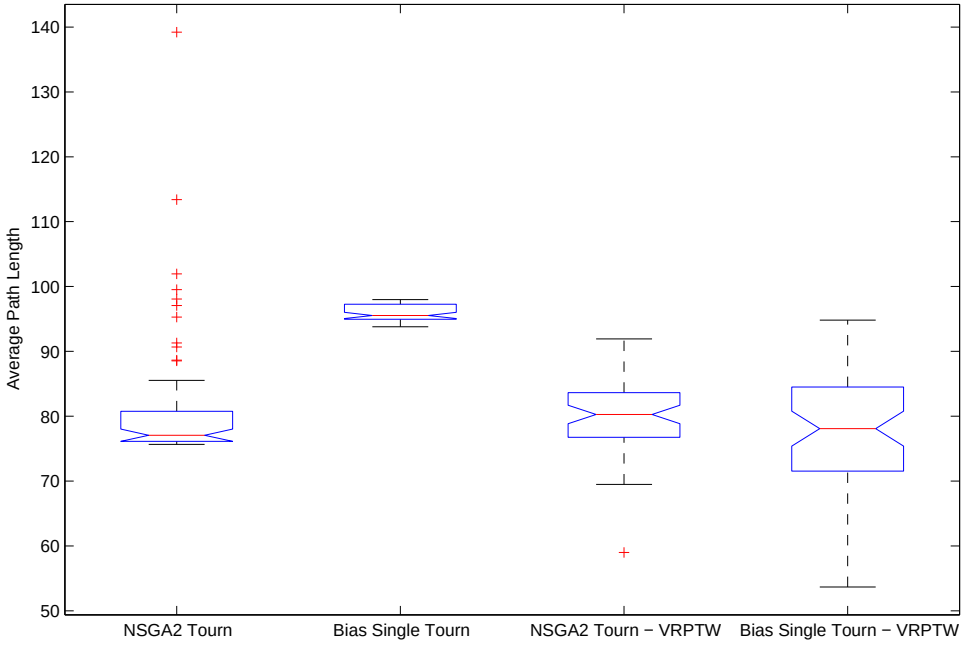

Figure 7.32: Comparison of average distance per vehicle in SRP results to VRPTW results for problem C103 with 25 customers

7.4 Comparative and Extended Analysis of Results

Results from the VRPTW experimentation showed a consistent return of results across a broad spectrum of problems. Optimization along the path length objective

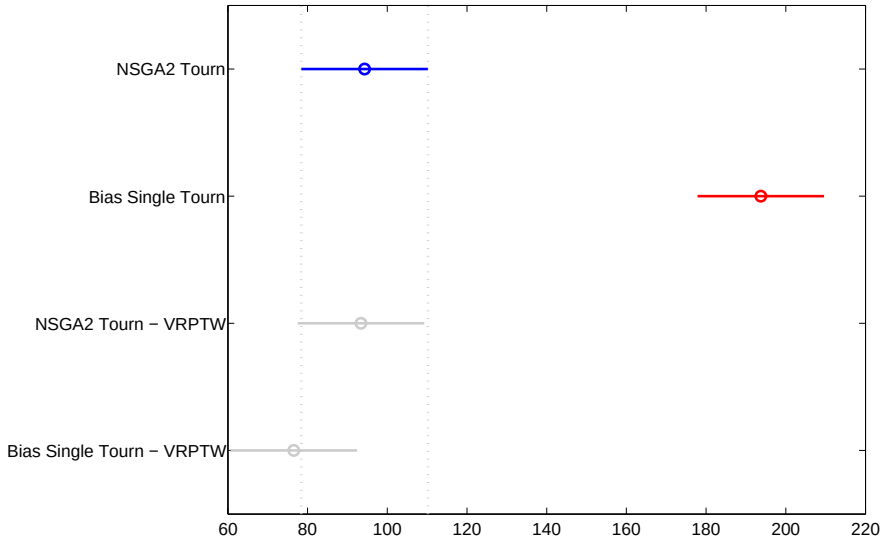

Figure 7.33: Statistical analysis comparing SRP results to VRPTW results for problem C103 with 25 customers

showed less than optimal results, however when combined with a view of the achieved non-dominated front, it is clear that the MOEA strategy is working correctly. Further comparison between results shows the NSGA2 algorithm performing better or as well as the biased algorithm. However, in some cases (Figure 7.14 and 7.21) the biased solution converged early while the MOEA approaches maintained a breadth of search in the solution space. These results lead to the conclusion that the MOEA solution method developed and implemented here is effective at optimization over a range of different problems.

Even with a multi-objective design, optimization of the path length objective still approaches optimal value. NSGA2 is able to achieve a path length value within 10 percent of the optimal value, and is even closer in some cases (Figure 7.9). It can be concluded that even while the algorithm is optimizing across multiple objectives the returns for a single objective are no being compromised, as evidenced by NSGA2s performance on the various benchmark problems.

With the validation of the MOEA design in place, obtained through analysis of results over VRPTW benchmark problems, attention can then be turned to to the SRP problem model. Results for the SRP again show consistent returns of total

and average path length. Average path length is then compared to the average path length returns for the VRPTW solution in order to show that the SRP achieves comparable results, which it does. That the average path length returns for the SRP are comparable indicates the merit of the model as a per vehicle optimization strategy. The purpose of the SRP as model is to develop time constrained routes between many different targets each of which requires some number of vehicle visitations. It is no sunrise that total path length is greater for the SRP returns, it would have to be, what is important is that the returned solution does not require any one vehicle to visit a large number of points, as would be the case in the VRPTW.

The benchmarks used in these experiments should also be considered reflective of real world problems and not merely contrived problems. A real world mission for a compliment of UAVs can conceivably contain 20 or more targets, to which these benchmarks affectively match. The SRP model shows capability not only as a combinatorics formulation but also as an applicable model for real world problem formulation, as the solutions shown here validate a capability to return consistent solutions.

7.5 Chapter Summary

This chapter presented the results and analysis from the VRPTW and SRP experiments defined in Chapter VI. Results from the VRPTW experimentation showed consistent returns across the spectrum of problems applied to it and results comparable to single objective optimization results found in the literature. Drawing from these results leads to the conclusion that the VRPTW algorithmic solution design constructed in this investigation represents a valid construct for optimizing VRPTW solutions.

Results from the SRP experiments showed consistent returns for the two algorithmic procedures examined. Average vehicle length results did not compare consistently favorably nor unfavorably to the VRPTW results. As the objectives for both problem models are arrived at through different procedures it should not be con-

cluded that the VRPTW model has outperformed the SRP. Rather, the results show the SRP model achieves per vehicle optimization comparable to the VRPTW meaning that while the number of vehicles used is greater, which would be an expected aspect of the models design, each vehicle does the same amount of average work. In the next chapter the objectives for this investigation are reviewed and future research is discussed.

VIII. Conclusions

Ruminations on the topics covered in this research lead to many different trains of thought. This chapter summarizes the results obtained and analyzed in the previous chapter and associates the objectives in the first chapter. A discussion of future research related to both UAV simulation and direct extensions of this work concludes the investigation.

8.1 Review of Accomplishments

This research endeavor has yielded a number of significant returns:

- Development of the **Swarm Routing Problem** (SRP) model is shown to be an **effective and solvable problem model for multi UAV routing**. The SRP evolves from the initial definition of the VRPTW and is modified it by **removing single target visitation restrictions** and changing **target satisfaction to be based on vehicle volume** at a certain point in time. These changes cause the model to **treat the vehicles as members of a sub-swarm** as opposed to discrete vehicles. The solution to this problem model **better represents reality** and offers an **optimized allocation of UAV resources**.
- Implementation of **MOEA solution in software using the Open Beagle evolutionary computation library**. The benefit of this library allowed for cross use of many methods between the VRPTW and SRP solution. The **software itself is also easily expandable and general** allowing for the **testing of a very large number of experimentation settings and algorithmic domains**.
- Results from the VRPTW experiments show **consistent returns often within and under 10 percent of published results**. The benefit of the solution presented is a **consistent return of results not constrained by problem specific information.**

- The proposed solution and SRP experimentation also shows the desired effect of **decreasing individual UAV path cost while still routing all UAVs to all targets within the time limits**.

8.1.1 Objective 1: Develop SRP as new model for UAV routing. The first objective, defined in Section 1.4.1, is to modify the existing VRPTW problem into a new type of combinatorics problem called the SRP. The drive from this originates from the VRPTW being an incomplete model for UAV routing. This stems from the VRPTW being a closer model to truck routing, where each vehicle is seen as a large movable entity capable of visiting many targets. This is not the case with UAV routing which consists of many small vehicles which satisfy target demand based not on inherent capacity but volume on location (the number of UAVs on site at a particular time). The VRPTW is modified to remove the restriction that each target be visited by only a single vehicle. The complexity of the problem remains at least as hard the VRPTW.

8.1.2 Objective 2: Develop and validate MOEA solution to VRPTW. The second objective uses an established combinatorics problem to model the routing of UAVs across multiple targets within time constraints. The problem is discussed in terms the multiple objectives of path length, vehicle count, and total wait time. A MOEA solution structure is chosen after analysis of the problem concludes the very fast growing solution space necessitates some form of stochastic solution. The MOEA solution is first designed in terms of a generic GA structure (defining the selection and replacement method) and then a definition of the specific operators used to modify individual solutions. Within this design phase an optimization strategy involving a local search technique is created. This local search takes place with a mutation operator called best route cost (see Section 4.8.5) where a single route is optimized across the parameter of path length.

The use of an MOEA necessitates the use of an evolutionary algorithm software package, resulting in the selection of the Open Beagle library written in C++. This

library offers a powerful set of tools for developing genetic algorithms without being problem specific. Genetic operators and an overall software structure are designed using this library for the VRPTW with great success.

Problems from the Solomon benchmark set are chosen in order to validate the solution and have comparable results to other achievements in the field. Results show effective solutions for 25 and and 50 dimension problems with large time windows with a degradation in performance for large dimension, highly constrained time window problems. As benchmark problems are optimized only for the single objective of path length defining algorithm effectiveness in terms of path length is ineffective, though it is used in order to have a comparable metric. Resultant solutions were within 10 percent of the benchmark value in most cases. The net effect of this experimentation shows the overall effectiveness of the MOEA routing method to develop "good" solutions to routing problems.

8.1.3 Objective 3: Develop and validate MOEA solution to SRP. As the complexity of the SRP is at least that of the VRPTW, shown in Section 4.2, a stochastic solution is deemed appropriate. The solution process parallels that of the VRPTW in terms of algorithm structure and the code library used. Experiments used the Solomon problems for the VRPTW. These problems were chosen due to there ease of availability and to have some type of result to compare to.

Since the structure of the SRP is different from the VRPTW exact solution comparison is not possible. The same objectives exist for both problems, however since targets are satisfied in different manners comparing objectives of path length and vehicle count are not appropriate. Instead, average path length per vehicle is used a comparison metric. Results showed that the proposed solution generated better results over a set number of iterations and that average path length was reduced. How much better the results are is arbitrary as the results generated for these benchmarks is unique to this work. The important point to take from this objective is that by

using the SRP problem model routing solutions can be developed which "optimize" the path for individual UAVs while still solving a large and complex routing problem.

8.2 Future Research and Closing Remarks

As mentioned in Chapter II this effort is one in a long line of research concerning UAV routing and simulation. With the development of this advanced routing capability, new developments in SO control schemes [32], and previous work developing a 3D UAV simulator the goal still remains of developing a high quality UAV swarm simulator (operational in either online or off line mode). Within the confines of this research next step investigations within UAV routing would be:

1. Further research on the SRP as a UAV routing problem model.
2. Integration of routing software with path planning software [44].
3. Full integration of mission planning software, recently developed SO controls [32], and the AFIT simulator [40]. Each of these pieces now exists and are ready to be combined.

Outside the realm of UAV routing the following topics could easily be extended from this work. These topics would extend understanding of the VRPTW, SRP, and MOEA solution techniques:

1. Development of different heuristic operators for the VRPTW in order to further optimize performance. Many local search techniques have already been defined for the VRPTW such as path savings strategies and problem relaxation techniques. It would take a small amount of effort to convert one of these search techniques into a mutation operator.
2. Alteration of design algorithm to handle other VRP variants to match different real world problem applications.

3. Expansion of MOEA solution method to handle very high dimension problems ($>$ 100 targets). Dealing with problem dimensions higher than 50 entails a number of complexity and decision problems.

4. Further research on the SRP as a combinatorics problems to define different solutions approaches and variants. Some possible solution approaches are described in Chapter 4.3.

Work yet remains in the field of UAV simulation and mission planning. The current pace of development for experiments such as the ones shown here be maintained or increased if significant information is to be obtained before hardware capable of meeting the demands of a real UAV swarm are developed. Currently at AFIT research within the Advanced Navigation Technology (ANT) Lab is focused on development of onboard sensor and computer systems that can provide a basis for further swarm development. The fastest development to a deploy-able UAV swarm is achieved when a full understanding and simulatable model of UAV swarms has been developed long before capable hardware is.

Appendix A. Evolutionary Computation

Evolutionary computation is a subfield within the domain of natural computing [8]. Natural computing an area of research in computer science that attempts to emulate and simulate processes found in nature for the purpose of solving real world problems. Evolutionary computation encompasses a wide number of search techniques. Throughout the literature the term GA is also used to signify an evolutionary algorithm. In the context of this work both terms refer to the same algorithm type.

An EC algorithm uses a process of manipulation in order to generate optimal solutions to single or multi-objective complex problems. This process (refereed to as evolving a solution) involves using methods of recombination, mutation, and selection on a population of solutions in order to develop this optimal solution [2]. This process is discussed in further detail in section A.1.

An important aspect of this evolutionary process is the structure of the chromosome that defines an individual solution. This structure is refereed to as a genotype and is what determines how the different processes in the GA occur (the actual solution is called the phenotype). The structure is also what determines how the solution is represented. Often the structure of the genotype determines how effective the GA is at finding better solutions. In this paper we present a new type of chromosome that possesses redundant information treated as a type of memory for that particular solution. By using this structure the algorithm is able to make more efficient use of the GA methods. This structure is incorporated into a GA and applied to a Traveling Salesman problem in order to illustrate the effect.

A.1 Classic Genetic Algorithm

A standard GA is defined mathematically as an evolutionary algorithm in [2]. The EA works via a process of selection, crossover, and mutation and is based on the chromosome structure the algorithm uses. For this paper we use a TSP problem to illustrate these functions within a GA.

A.1.1 Chromosome Structure. The chromosome structure (genotype) is what determines the solution to the problem being solved (phenotype). The TSP is a combinatorics problem that easily lends itself to being solved with a Genetic Algorithm due to the way a solution can be structured. There are many variations to the classic TSP, however in its most general form it is a graph consisting of n vertexes and n-1 edges. The goal is to determine a path from one vertex, through all the other vertex points exactly once and ending back at the start point (called a circuit). The objective of the problem is to determine the shortest possible path that accomplishes this goal. This is illustrated in Figure A.1 though the graph edges are not shown. It is assumed that distances between cities is Euclidean (i.e. the distance from 1 to 2 is less than the distance from 1 to 5).

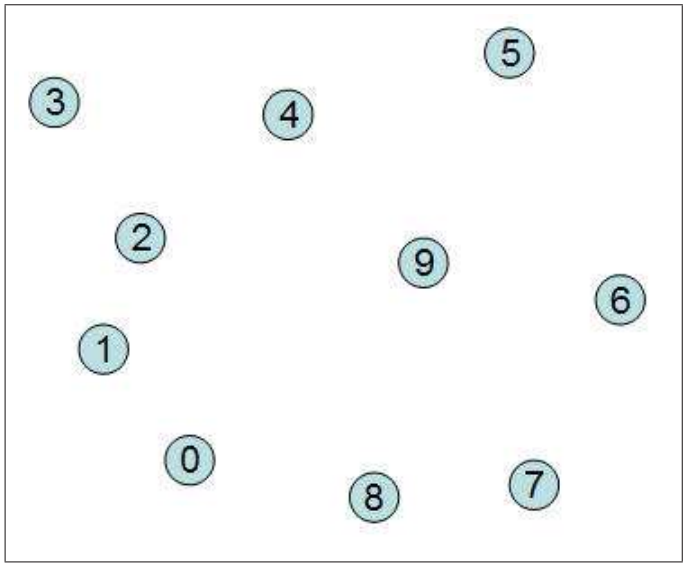

Figure A.1: An Unsolved TSP

The chromosome structure is a code that defines a particular solution. In [27] a TSP genotype is formatted as follows. All cities in the graph are viewed as an index. Each city has a connection point that indicates which city it leads to. This chromosome structure as well as the resultant TSP solution is illustrated in Figure A.2. This structure is used to facilitate crossover operations that are discussed in the next section.

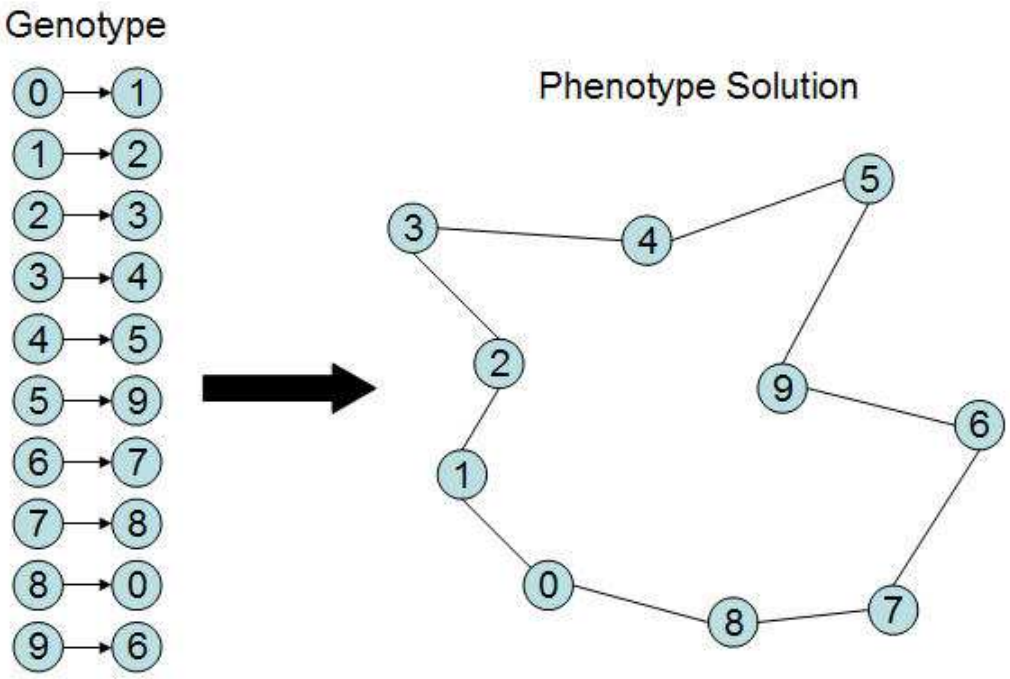

Figure A.2: A TSP Genotype and Solution

Please note that the structure illustrated is not the exact structure that would be used in a coded implementation. Obviously the information indicating the index is redundant and is only shown for illustrative purposes. The only necessary information is the second string where the index is implied by the order of the vertexes.

A.1.2 Crossover, Mutation, and Selection. Crossover, mutation, and selection are the methods the GA uses that provide the search functionality. Each can be constructed in a variety of ways and each impacts the effectiveness of the GA search process. We continue to use the TSP as the example problem.

Crossover is a process where two solutions are "crossbred" resulting in two children genotypes, each of which is constructed with some aspect of its parent. This process is illustrated in Figure A.3.

Note that the crossover operation is not a static process. Often times crossover results in an invalid solution. For example when Parent II tries to add its bottom

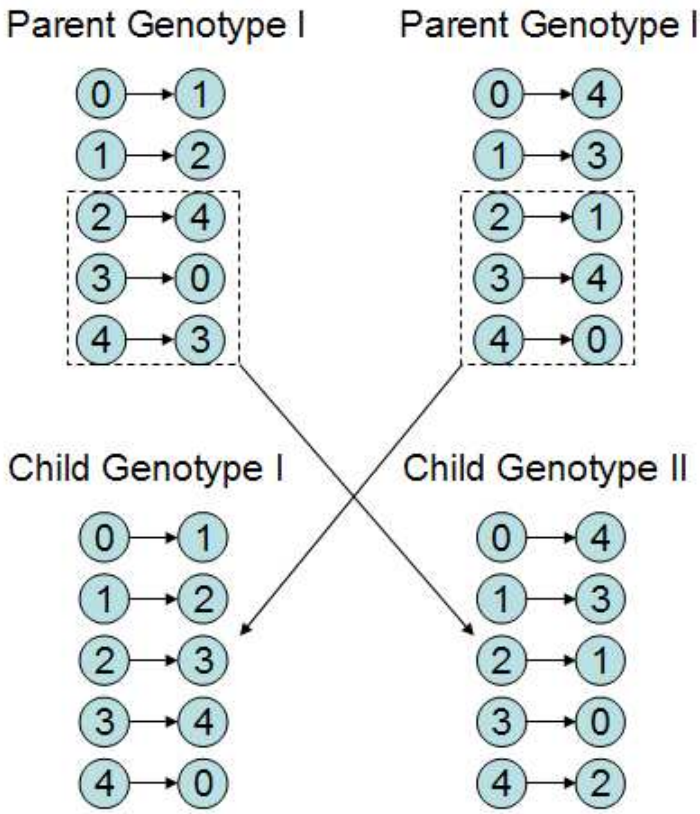

Figure A.3: An example of Crossover

three vertexes to Child I there is a conflict if vertex 2 were to point to vertex 1, because vertex 1 already points to vertex 2. This problem can be addressed in two ways. Either the invalid solution can be dropped from the population or the solution can be repaired. This repair result is also shown in Figure A.3.

If only crossover operations occurred no new information would ever be added to the system. Thus after a few generations most of the genotypes would start to look the same (depending on the selection process used). This is the classic struggle of exploration versus exploitation. In order to add new information to the system we incorporate the concept of mutation. Mutation is a random change applied stochastically to a genotype. The result of this operation is a random tour that is incorporated into the population at a random interval. An example of this process is shown in Fig-

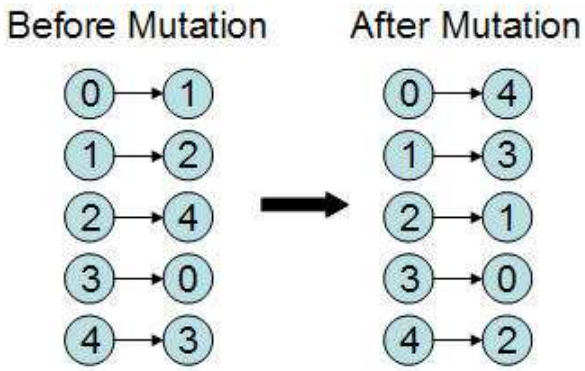

Figure A.4: An example of Mutation

ure A.4. Note that the net effect of mutation is actually just a switch of two vertex points.

As new population members are created via Crossover and Mutation a process must exist that eliminates ineffective solutions. In evolutionary computation a property called fitness is used to describe how effective a solution is. In the case of our single objective TSP the fitness of a solution is the total length of the circuit. This concept of fitness combined with a deterministic or stochastic selection method determines which of the new population members continue into future generations. A detailed examination of this method and the others discussed in this section can be found in [7]. These operations are incorporated into a pseudo code description of an EA in Algorithm 6.

Algorithm 6 Generic EA

1: **procedure** EA
2: $t := 0$;
3: *generate* $P(0) := \{\vec{a}_1(0), ..., \vec{a}_\mu(0)\} \in I^\mu$;
4: *evaluate* $P(0) : \{\Phi(\vec{a}_1(0)), ..., \Phi(\vec{a}_\mu(0))\}$;
5: **while** $(\iota(P(t) \neq \mathit{true})$ **do**
6: *recombine*: $P'(t) := r\Theta_r(P(t))$;
7: *mutate*: $P''(t) := r\Theta_r(P'(t))$;
8: *evaluate*: $P''(t) : \Phi(\vec{a}''_1(t)), ..., \Phi(\vec{a}''_\mu(t))$;
9: *select*: $P(t+1) = s\Theta_s(P''(t) \bigcup Q)$;
10: $t := t + 1$;
11: **end while**
12: **end procedure**

Appendix B. Implimentation Documentation

The software library selected for this investigation is the Open Beagle Evolutionary Computation library [18]. The library consists of a complete evolutionary computation infrastructure coded in object oriented C++. This library has a number of benefits over coding an evolutionary algorithm for a specific case.

The software is structured such that each component of an evolutionary algorithm is represented within its own class structure. Each structure is then controlled and managed by an overarching control structure that manages the application of each operation, the population, and statistical analysis. Control of all the operations is managed by a user created XML file. This file controls the structure of the evolutionary algorithm to be used, statistical tools to be applied, and how results should be stored. Settings for the algorithm can either be set within the XML sheet or on the command line.

Open Beagle also offers integrated printing of debug statements. Rather than using standard output methods open beagle uses its own structure for controlling logged output. This allows for a hierarchy of debug information. By setting each debug setting to a level appropriate to its information content total output information can then be set from the command line.

Open Beagle offers an infinite expendability and modification capability within the limits of standard GA definitions. By formulating each class in only general terms and wrapping the individuals within a standardized class the implementation of any GA design is allowed take place without being constrained by the Open Beagle architecture. In regards to the implementation discussed in this document four important aspects were created. The customer class, the chromosome data structure, and various genetic operators. The construction concept of one of the operators is found in Chapter V.

B.1 The Customer Class

The customer class is created as a persistent object at the intimidation phase. This object then creates and stores all the problem relevant data. Each operator that requires the use of a customer class method is passed a reference pointer to the customer class object. In this way all problem information and evaluation capability is kept within the customer class. This serves the purpose of simplifying the evaluation of individuals and keeping information in a single location as apposed to being passed among the population. Each population member is only a code of numbers whose meaning exists in the customer class.

All methods within the GA also call the customer class in order to evaluate solutions. This not only eliminates the need to pass problem information between classes it also creates a single evaluation function that entire algorithm uses simplifying the construction process and adhering to basic software engineering standards.

B.2 Chromosome Data Structure

The chromosome data structure had to be created as a standard one did not exist with the Open Beagle library. The design required a data structure consisting of a vector of integer vectors. To accomplish this the integer vector data structure within the Open Beagle was modified such that the vector contained integer vectors instead of just integers. A new print function was also created for displaying the chromosome in the output files.

Bibliography

1. Agarwal, A., M.-H. Lim, M.Y.W. Kyaw, and M.J. Er. "Inflight rerouting for an unmanned aerial vehicle". *Genetic and Evolutionary Computation - GECCO 2004. Genetic and Evolutionary Computation Conference. Proceedings, Part II (Lecture Notes in Computer Science Vol.3103)*, p859 – 868. Springer-Verlag, Intelligent Syst. Center, Nanyang Technol. Univ., Singapore, 2004. ISBN 3 540 22343 6. URL `http://search.ebscohost.com/`.

2. Bäck, Thomas. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford University Press, Oxford, UK, 1996. ISBN 0-19-509971-0.

3. Barn, Benjamn and Matilde Schaerer. "A Multiobjective Ant Colony System for Vehicle Routing Problem with Time Windows". *Proceedings of the 21st IASTED International Conference.* Innsbruck, Austria, February 2003. URL `http://www.actapress.com/PaperInfo.aspx?PaperID=14090`.

4. Benyahia, I. and J.-Y. Potvin. "Decision support for vehicle dispatching using genetic programming". *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 28(3):306–314, May 1998.

5. Bleuler, Stefan, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. "PISA — A Platform and Programming Language Independent Interface for Search Algorithms". Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele (editors), *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, 494 – 508. Springer, Berlin, 2003.

6. Castillo, Oscar, Leonardo Trujillo, and Patricia Melin. "Multiple Objective Genetic Algorithms for Path-planning Optimization in Autonomous Mobile Robots". *Soft Comput.*, 11(3):269–279, 2006. ISSN 1432-7643.

7. de Castro, Leandro Nunes. *Fundamentals of Natural Computing (Chapman & Hall/Crc Computer and Information Sciences).* Chapman & Hall/CRC, 2006. ISBN 1584886439.

8. de Castro, Leandro Nunes and Fernando Jos Von Zuben. *Artificial Immune Systems: Part Ii - A Survey Of Applications.* Technical Report DCA-RT 02/00, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, SP, Brazil, Feb 2000. URL `citeseer.ist.psu.edu/nunesdecastro00artificial.html`.

9. Coello Coello, Carlos A. and Gary B. Lamont (editors). *Application of Multi Oobjective Evolutionary Algorithms*, volume 1 of *Advances in Natural Computation.* World Scientific Publishing, 2004.

10. Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001. ISBN 0262531968. URL `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0262531968`.

11. Corner, Joshua. *Swarming Reconnaissance Using Unmanned Aerial Vehicles In A Parallel Discrete Event Simulation*. Master's thesis, Air Force Institute of Technology, 2004.

12. Daz, Bernab Dorronsoro. "The VRP Web: Solution Techniques for VRP". `http://neo.lcc.uma.es/radi-aeb/WebVRP/m`, 2007. URL `http://neo.lcc.uma.es/radi-aeb/WebVRP/`.

13. Deb, K., L. Thiele, M. Laumanns, and E. Zitzler. "Scalable Test Problems for Evolutionary Multi-Objective Optimization". A. Abraham, R. Jain, and R. Goldberg (editors), *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*, chapter 6, 105–145. Springer, 2005. ISBN 1-85233-787-7.

14. Deb, Kalyanmoy, Samir Agrawal, Amrit Pratab, and T. Meyarivan. "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II". Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, J. J. Merelo, and Hans-Paul Schwefel (editors), *Proceedings of the Parallel Problem Solving from Nature VI Conference*, 849–858. Springer. Lecture Notes in Computer Science No. 1917, Paris, France, 2000. URL `citeseer.ist.psu.edu/deb00fast.html`.

15. Dogan, A. "Probabilistic approach in path planning for UAVs". *Intelligent Control. 2003 IEEE International Symposium on*, 608–613, 2003.

16. Donati, A., L. Gambardella, N. Casagrande, A. Rizzoli, and R. Montemanni. "Time Dependent Vehicle Routing Problem with an Ant Colony System". URL `citeseer.ist.psu.edu/562261.html`.

17. Gagn, Christian. "Documentation:Manual:Architecture". Online, http://beagle.sourceforge.net/wiki/index.php/Documentation:Manual:Architecture, 2007.

18. Gagn, Christian and Julie Beaulieu. "Open BEAGLE W3 Page". `http://beagle.gel.ulaval.ca/index.html`, 2006. URL `http://beagle.gel.ulaval.ca/index.html`.

19. Gambardella, L. M., E. Taillard, and G. Agazzi. *MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows*. Technical report, Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, 1999.

20. Gerkey, Brian Paul. *On multi-robot task allocation*. Ph.D. thesis, Los Angeles, CA, USA, 2003. Adviser-Maja J. Mataric.

21. Homberger, Joerg and Hermann Gehring. "Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows". *INFOR Journal*, 37(3):297 – 318, August 1999.

22. Hu, Xiaohui, Russell C. Eberhart, and Yuhui Shi. "Particle Swarm with Extended Memory for Multiobjective Optimization". *Swarm Intelligence Symposium, 2003. SIS '03. Proceedings of the 2003 IEEE*, 193 – 197. April 2003. URL `citeseer.ist.psu.edu/hu03particle.html`.

23. Kadrovach, Tony. *Communications Modeling System For Swarm-Based Sensors*. Ph.D. thesis, Air Force Institute of Technology, 2003.

24. Kleeman, Mark. *Evaluation and Optimization of UAV Swarm Multi-Objectives*. Master's thesis, Air Force Institute of Technology, 2004.

25. Kostaras, A., I. Nikolos, and K. Valavanis. "Evolutionary Algorithm based 3-D On-Line Path Planner for UAV Navigation". K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty (editors), *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, 481–486. International Center for Numerical Methods in Engineering (Cmine), Athens, Greece, 2001. ISBN 84-89925-97-6.

26. Ladd, A. M. and L. E. Kavraki. "Fast Tree-Based Exploration of State Space for Robots with Dynamics". M. Erdmann, D. Hsu, M. Overmars, and A. F. van der Stappen (editors), *Algorithmic Foundations of Robotics VI*, 297–312. Springer, STAR 17, 2005.

27. LaLena, Michael. "Traveling Salesman Problem Using Genetic Algorithms". `http://www.lalena.com/AI/Tsp/`, 2006. URL `http://www.lalena.com/AI/Tsp/`.

28. Lotspeich, James. *Distributed Control of a Swarm of Autonomous Unmanned Aerial Vehicles*. Master's thesis, Air Force Institute of Technology, 2003.

29. Lou, Shan-Zuo and Zhong-Ke Shi. "A New Method for Multi-Depot Vehicle Routing Problem with Time Windows". *Machine Learning and Cybernetics, 2006 International Conference on*, 2503–2509. Aug. 2006.

30. Machado, Penousal, Jorge Tavares, Francisco B. Pereira, and Ernesto Costa. "Vehicle Routing Problem: Doing It The Evolutionary Way". *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, 690. Morgan Kaufmann Publishers, New York, 9-13 July 2002. ISBN 1-55860-878-8. URL `citeseer.ist.psu.edu/machado02vehicle.html`.

31. Milam, Keven. *Evolution of Control Programs for a Swarm of Autonomous Unmanned Aerial Vehciles*. Master's thesis, Air Force Institute of Technology, 2004.

32. Nowak, Dustin. *EXPLOITATION OF SELF ORGANIZATION IN UAV SWARMS FOR OPTIMIZATION IN COMBAT ENVIRONMENTS*. Master's

thesis, Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 2008.

33. Ombuki, Beatrice, Brian J. Ross, and Franklin Hanshar. "Multi-Objective Genetic Algorithms for Vehicle Routing Problem with Time Windows". *Applied Intelligence*, 24(1):17–30, 2006. ISSN 0924-669X.

34. Pohl, Adam. "Applying Evolutionary Algorithms to the Vehicle Routing and Path Planning Problem", 2007. Assignment for CSCE 886.

35. Pohl, Adam. "An Examination of SPEA2 Performance Using PISA", 2007. Assignment for CSCE 886.

36. Price, Ian C. *Evolving Self Organizing Behavior for Homogeneous and Heterogeneous Swarms of UAVs and UCAVs*. Master's thesis, Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 2006.

37. Reynolds, C. W. "Interaction with Groups of Autonomous Characters". *Game Developers Conference*. 2000.

38. Reynolds, Craig W. "Flocks, Herds, and Schools: A Distributed Behavioral Model". *Computer Graphics*, 21(4):25–34, 1987. URL `citeseer.ist.psu.edu/reynolds87flocks.html`.

39. Rothlauf, Franz and David E. Goldberg. *Representations for Genetic and Evolutionary Algorithms*. Physica-Verlag, 2002. ISBN 3790814962.

40. Russell, M.A., G.B. Lamont, and K. Melendez. "On using SPEEDES as a platform for a parallel swarm simulation". *Winter Simulation Conference, 2005 Proceedings of the*, 9pp. 4-7 Dec. 2005.

41. Russell, Matthew A. and Gary B. Lamont. "A genetic algorithm for unmanned aerial vehicle routing". *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, 1523–1530. ACM Press, New York, NY, USA, 2005. ISBN 1-59593-010-8.

42. Secrest, Barry. *Traveling Salesman Problem for Surveillance Mission Using Particle Swarm Optimization*. Master's thesis, Air Force Institute of Technology, 2001.

43. Sezer, Ergin. *Mission Route Planning with Multiple Aircraft & Targets Using Parallel A* Algorithm*. Master's thesis, Air Force Institute of Technology, 2000. URL `http://www.stormingmedia.us/11/1190/A119083.html`.

44. Slear, James N. *AFIT UAV Swarm Mission Planning and Simulation System*. Master's thesis, Air Force Institute of Technology, 2006.

45. Solomon, M. M. "Algorithms for the vehicle routing and scheduling problems with time window constraints". *Oper. Res.*, 35(2):254–265, 1987. ISSN 0030-364X.

46. Sugihara, Kazuo. "A Genetic Algorithm for 3-D Path Planning of a Mobile Robot". URL citeseer.ist.psu.edu/177344.html.

47. Tan, K., L. Lee, Q. Zhu, and K. Ou. "Heuristic methods for vehicle routing problem with time windows". *Artificial Intelligence in Engineering*, 15:281–295, 2001. URL citeseer.ist.psu.edu/article/tan99heuristic.html.

48. Tan, K.C., T.H. Lee, Y.H. Chew, and L.H. Lee. "A multiobjective evolutionary algorithm for solving vehicle routing problem with time windows". *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 1, 361–366vol.1. 5-8 Oct. 2003.

49. Tavares, Jorge, Penousal Machado, Francisco B. Pereira, and Ernesto Costa. "On the influence of GVR in vehicle routing". *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, 753–758. ACM Press, New York, NY, USA, 2003. ISBN 1-58113-624-2.

50. Tavares, Jorge, Francisco B. Pereira, Penousal Machado, and Ernesto Costa. "GVR Delivers It On Time". *4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'02)*, 745–749. 2002. URL http://cisuc.dei.uc.pt/ecos/dlfile.php?fn=61_pub_vrp_seal2002.pdf&get=1&idp=61&ext=.

51. Toth, Paolo and Daniele Vigo (editors). *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-498-2.

52. Unknown. "Raven UAV Draws Raves From The Field". Online, Feb 2005. URL http://www.defenseindustrydaily.com/raven-uav-draws-raves-from-the-field-067.

53. Walls, Matthew. "GAlib: A C++ Library of Genetic Algorithm Components". URL http://lancet.mit.edu/ga/.

54. Watson, N.R., N.W. John, and W.J. Crowther. "Simulation of unmanned air vehicle flocking". *Theory and Practice of Computer Graphics*, 3:130–137, 2003. URL http://ieeexplore.ieee.org/xpls/.

55. Weisstein, Eric W. "Partition Function P." From MathWorld–A Wolfram Web Resource. URL http://mathworld.wolfram.com/PartitionFunctionP.html.

56. Wikipedia. "Wikipedia, the free encyclopedia: Pareto Efficiency". http://en.wikipedia.org/wiki/Pareto_efficiency, 2007. URL http://en.wikipedia.org/wiki/Pareto_efficiency.

57. Xiao, Jing, Zbigniew Michalewicz, and Krzysztof Trojanowski. "Adaptive Evolutionary Planner/Navigator for Mobile Robots". *IEEE Transactions on Evolutionary Computation*, 1(1):18–28, 1997.

58. Yanwei, Zhao, Wu Bin, Ma Yaliang, Dong Hongzhao, and Wang Weian. "Particle Swarm Optimization method for Vehicle Routing Problem". *Intelligent Control and Automation*, 3:2219–2221, 2004.

59. Yavuz, Ekursat. *Multiobjective Mission Route Planning Using Particle Swarm Optimization*. Master's thesis, Air Force Institute of Technology, 2002. URL `http://stinet.dtic.mil/oai/`.

60. Zhu, Kenny Qili. "A New Genetic Algorithm For VRPTW". *International Conference on Artificial Intelligence, Las Vegas, USA*. 2000.

61. Zhu, Qing, Limin Qian, Yingchun Li, and Shanjun Zhu. "An Improved Particle Swarm Optimization Algorithm for Vehicle Routing Problem with Time Windows". *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 1386–1390. 16-21 July 2006.

62. Zitzler, E., K. Deb, and L. Thiele. "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results". *Evolutionary Computation*, 8(2):173–195, 2000.

63. Zitzler, Eckart, Dimo Brockhoff, and Lothar Thiele. "The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted Integration." Shigeru Obayashi, Kalyanmoy Deb, Carlo Poloni, Tomoyuki Hiroyasu, and Tadahiko Murata (editors), *EMO*, volume 4403 of *Lecture Notes in Computer Science*, 862–876. Springer, 2006. ISBN 3-540-70927-4.

64. Zitzler, Eckart, Marco Laumanns, and Lothar Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 2001.

65. Zitzler, L., E.; Thiele. "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach". *Evolutionary Computation, IEEE Transactions on*, 3(4):257–271, Nov 1999. ISSN 1089-778X.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
27-03-2008	**Master's Thesis**	Aug 2006 – Mar 2008

4. TITLE AND SUBTITLE

Multi-Objective UAV Mission Planning Using Evolutionary Computation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Pohl, Adam J., 2LT, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENG)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GE/ENG/08-22

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFRL/SNZW
Attn: Mike Foster michael.foster@wpafb.af.mil
2241Avionics Circle
WPAFB OH 45433-7303 DSN: 986-4899x3030

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This investigation purports to develop a new model for multiple autonomous aircraft mission routing. Previous research both related and unrelated to this endeavor have used classic combinatoric problems as models for \ac{UAV} routing and mission planning. This document presents the concept of the Swarm Routing Problem (SRP) as a new combinatorics problem for use in modeling UAV swarm routing, developed as a variant of the Vehicle Routing Problem with Time Windows (VRPTW). The SRP removes the single vehicle per target restraint and changes the customer satisfaction requirement to one of vehicle on location volume. The impact of these alterations changes the vehicle definitions within the problem model from discrete units to cooperative members within a swarm. This represents a more realistic model for multi-agent routing as a real world mission plan would require the use of all airborne assets across multiple targets, without constraining a single vehicle to a single target. Solutions to the SRP problem model result in route assignments per vehicle that successfully track to all targets, on time, within distance constraints. A complexity analysis and multi-objective formulation of the VRPTW indicates the necessity of a stochastic solution approach leading to the development of a multi-objective evolutionary algorithm. This algorithm design is implemented using C++ and an evolutionary algorithm library called Open Beagle. Benchmark problems applied to the VRPTW show the usefulness of this solution approach. A full problem definition of the SRP as well as a multi-objective formulation parallels that of the VRPTW method. Benchmark problems for the VRPTW are modified in order to create SRP benchmarks. These solutions show the SRP solution is comparable or better than the same VRPTW solutions, while also representing a more realistic UAV swarm routing solution.

15. SUBJECT TERMS
Unmanned Aerial Vehicles, Genetic Algorithms, Evolutionary Computation, Vehicle Routing Problem with Time Windows

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
REPORT	**ABSTRACT**	**c. THIS PAGE**			**Dr. Gary B. Lamont (AFIT/ENG)**
U	**U**	**U**	**UU**	135	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636 x 4718; Email; gary.lamont@afit.edu